Exact Solution of the Quadratic Knapsack Problem

Article *in* Informs Journal on Computing · October 1998

DOI: 10.1287/ijoc.11.2.125 · Source: CiteSeer

3 authors, including:

David Pisinger
Technical University of Denmark
163 PUBLICATIONS **10,142** CITATIONS

SEE PROFILE

Paolo Toth
University of Bologna
236 PUBLICATIONS **19,733** CITATIONS

SEE PROFILE

Exact Solution of the Quadratic Knapsack Problem

Alberto Caprara[1], David Pisinger[2], Paolo Toth[1]
[1]DEIS, Univ. of Bologna, Viale Risorgimento 2, Bologna, Italy
e-mail: {acaprara,ptoth}@deis.unibo.it
[2]DIKU, Univ. of Copenhagen, Univ.parken 1, Copenhagen, Denmark
e-mail: pisinger@diku.dk

December 1997, revised July 1998

Abstract

The *Quadratic Knapsack Problem* (QKP) calls for maximizing a quadratic objective function subject to a knapsack constraint, where all coefficients are assumed to be nonnegative and all variables are binary. The problem has applications in location and hydrology, and generalizes the problem of checking whether a graph contains a clique of a given size.

We propose an exact branch-and-bound algorithm for QKP, where upper bounds are computed by considering a Lagrangian relaxation which is solvable through a number of (continuous) knapsack problems. Suboptimal Lagrangian multipliers are derived by using subgradient optimization and provide a convenient reformulation of the problem. We also discuss the relationship between our relaxation and other relaxations presented in the literature. Heuristics, reductions and branching schemes are finally described. In particular, the processing of each node of the branching tree is quite fast: We do not update the Lagrangian multipliers, and use suitable data structures to compute an upper bound in linear expected time in the number of variables.

We report exact solution of instances with up to 400 binary variables, i.e., significantly larger than those solvable by the previous approaches. The key point of this improvement is that the upper bounds we obtain are typically within 1% of the optimum, but can still be derived effectively. We also show that our algorithm is capable of solving reasonable-size Max Clique instances from the literature.
Keywords: 0-1 Quadratic Programming, Knapsack Problem, Lagrangian Relaxation, Branch-and-Bound.

Introduction

We are given n items, the j-th having a positive integer weight w_j, a positive integer knapsack capacity c and an $n \times n$ nonnegative integer matrix $P = (p_{ij})$, where p_{jj} is a profit achieved if item j is selected, and, for $j > i$, $p_{ij} + p_{ji}$ is a profit achieved if both items i and j are selected. The *Quadratic Knapsack Problem* (QKP) calls for selecting an item subset whose overall weight does not exceed the knapsack capacity, so as to maximize the overall profit. For notational convenience, let $N := \{1, \ldots, n\}$ denote the item set and $q_j := p_{jj}$ denote the diagonal elements of P. By introducing a binary variable x_j equal to 1 if item j is selected and 0 otherwise, the problem has the following mathematical formulation:

$$\begin{aligned} \text{maximize} \quad & z(\text{QKP}) = \sum_{i \in N} \sum_{j \in N} p_{ij} x_i x_j \\ \text{subject to} \quad & \sum_{j \in N} w_j x_j \leq c \\ & x_j \in \{0, 1\}, \quad j \in N. \end{aligned} \tag{1}$$

We assume without loss of generality that $\max_{j \in N} w_j \leq c < \sum_{j \in N} w_j$ and that the profit matrix is symmetric, i.e., $p_{ij} = p_{ji}$ for all $i, j \in N, j > i$.

QKP is a generalization of the *Knapsack Problem* (KP), which arises when $p_{ij} = 0$ for all $i \neq j$. Moreover, QKP has the following immediate graph-theoretic interpretation. Given a complete undirected graph on node set N, where each node j has a profit q_j and weight w_j and each edge (i, j) has a profit $p_{ij} + p_{ji}$, select a node subset $S \subseteq N$ whose overall weight does not exceed c so as to maximize the overall profit, given by the sum of the profits of the nodes in S and of the edges with both endpoints in S. It is then easy to see that QKP is also a generalization of the *Clique* problem. This latter problem, in its recognition version, calls for checking whether, for a given positive integer k, a given undirected graph $G = (V, E)$ contains a complete subgraph on k nodes. A possible optimization version of Clique is given by the so-called *Dense Subgraph Problem*, in which one wants to select a node subset $K \subseteq V$ of cardinality $|K| = k$ such that the subgraph of G induced by K contains as many edges as possible. This problem can be modeled as (1) by setting $n := |V|$; $c := k$; $w_j := 1$ for $j \in N$; $p_{ij} := p_{ji} := 1$ if $(i, j) \in E$ and $p_{ij} := p_{ji} := 0$ otherwise, for $i, j \in N$. Note that in this case the knapsack constraint reduces to a cardinality constraint, and will be satisfied with equality by the optimal solution. Clearly, the answer to Clique is positive if and only if the optimal solution of this QKP has value $k(k-1)$. The most famous optimization version of Clique, called *Max Clique*, calls for an induced complete subgraph with a maximum number of nodes. This latter problem can be solved through a QKP algorithm by using binary search.

Max Clique, besides being (strongly) NP-hard, is one of the hardest combinatorial optimization problems studied in the literature, both from a theoretical approximability and from a practical solvability point of view. The same properties apply therefore to QKP as well, which is consequently much more difficult than the classical KP. In particular finding an approximate QKP solution of value not smaller than the optimum divided by

n^ϵ is NP-hard for any $\epsilon < \frac{1}{4}$ [5]. In view of these results, one should expect that any upper bound which can be computed efficiently will be extremely bad for some instances.

QKP was first studied by Gallo, Hammer and Simeone [13], who proposed exact algorithms where upper bounds are computed by using *upper planes*, which are linear functions of the binary variables which are not smaller than the QKP objective function over the set of feasible QKP solutions. Apparently, the problem was not widely studied until a few years ago, but has recently attracted great interest. Billionnet and Calmels [6] follow a branch-and-cut approach to the problem, using a classical ILP formulation with $O(n^2)$ variables and constraints. Lagrangian relaxation approaches are described by Chaillou, Hansen and Mahieu [9], Michelon and Veilleux [24], Hammer and Rader [14] and Billionnet, Faye and Soutif [7]. Helmberg, Rendl and Weismantel [17] consider a more general version of the problem where P may have negative entries, and propose a combined approach which uses cutting planes and semidefinite programming, and allows for the computation of very tight upper bounds. The *Integer* QKP, where variables may take any integer value between a lower and an upper bound, is considered by Bretthauer, Shetty and Syam [8], however restricted to *diagonal* profit matrices P, such that $p_{ij} = 0$ for $i \neq j$. Note that the general integer QKP can easily be formulated as a QKP by applying the same transformation as that from the *Bounded* KP to the KP described by Martello and Toth [21].

As one might expect, due to its generality, QKP has a wide spectrum of applications. Witzgall [27] presented a problem which arises in telecommunications when a number of sites for satellite stations have to be selected, such that the global traffic between these stations is maximized and a budget constraint is respected. This problem appears to be a QKP. Similar models arise when considering the location of airports, railway stations or freight handling terminals [26]. Johnson, Mehrotra and Nemhauser [18] mention a compiler design problem which may be formulated as a QKP, as described in [17]. Dijkhuizen and Faigle [11] and Park, Lee and Park [25] consider the weighted maximum b-clique problem. If all edge weights are nonnegative this problem is the special case of QKP arising when $w_j = 1$ for $j \in N$ and $b = c$. Finally, QKP appears as the column generation subproblem when solving the graph partitioning problem described in Johnson, Mehrotra and Nemhauser [18].

In this paper we propose an exact branch-and-bound algorithm for QKP, where upper bounds are computed by considering a Lagrangian relaxation which is solvable through a number of (continuous) KPs. Suboptimal Lagrangian multipliers are derived by using subgradient optimization and provide a convenient reformulation of the problem. We also discuss the relationship between our relaxation and other relaxations presented in the literature. Heuristics, reductions and branching schemes are finally described. In particular, the processing of each node of the branching tree is quite fast: We do not update the Lagrangian multipliers, and use suitable data structures to compute an upper bound in linear expected time in the number of variables.

We report the exact solution of instances with profit matrices up to 400×400 whereas the largest instances solved in the literature have size 100×100. The key point of this improvement is that the upper bounds we obtain are typically within 1% of the optimum, but can still be derived effectively. We also show that our algorithm is capable of solving

reasonable-size Max Clique instances from the literature.

We stress that some parts of our algorithm rely on the (usual) assumption that the profit matrix has nonnegative entries, which is not made without loss of generality. In particular, the well-studied *Max Cut Problem* and the related *0-1 Quadratic Programming Problem* are *not* trivial special cases of QKP.

The paper is organized as follows. In the following section we show how tight upper bounds can be derived through Lagrangian relaxation. The relaxed problem calls for the solution of a number of continuous KPs. The overall branch-and-bound algorithm is presented in Section 2, where we describe heuristics, reduction procedures, branching schemes and parametric computation of upper bounds in linear expected time. Finally, extensive computational experiments are reported in Section 3.

1 Effective Computation of a Tight Upper Bound

The choice of upper bounding procedures to be used in a branch-and-bound scheme for the solution of a maximization problem is usually based on a tradeoff between the tightness of the bound obtained and the time required for its computation. Depending on the particular problem at hand, different policies may be worth using. For the QKP instances in the literature, we found that a sufficiently tight upper bound can be computed in a relatively short time by effective combinatorial algorithms which avoid the use of general-purpose linear programming solvers as in [6], or, even more cumbersome, semidefinite programming solvers as in [17]. Our upper bound is based on a fast (dual heuristic) solution of a linear programming relaxation which is similar to others presented in the literature. A main contribution is a convenient reformulation of the problem obtained from the upper bound computation. This reformulation is effectively used within the branch-and-bound algorithm presented in the next section.

In our upper bounding procedure, we first add to formulation (1) some constraints which are redundant as long as the integer restriction on the variables is imposed, but tighten the continuous relaxation obtained by replacing, for $j \in N$, the constraint $x_j \in \{0,1\}$ with $0 \leq x_j \leq 1$. These new constraints are explicitly used in [17] and [6] and implicitly considered (as we will show) in [13]. For $j \in N$, we multiply the knapsack constraint by x_j and replace x_j^2 by x_j, getting the valid inequalities

$$\sum_{i \in N \setminus \{j\}} w_i x_i x_j \leq (c - w_j) x_j, \quad j \in N.$$

These constraints are part of the constraints which can be obtained by applying a general procedure proposed by Adams and Sherali [1] and further studied by Lovász and Schrijver [20] and Balas, Ceria and Cornuéjols [2].

Then, in order to linearize the formulation, we introduce a binary variable y_{ij} for $i, j \in N, j \neq i$ which replaces the product $x_i x_j$ in the formulation. These new variables are linked to the old ones by suitable inequalities, obtaining the *Integer Linear Programming*

(ILP) reformulation:

$$\text{maximize} \quad z(\text{QKP}) = \sum_{j \in N} \sum_{i \in N \setminus \{j\}} p_{ij} y_{ij} + \sum_{j \in N} q_j x_j \tag{2}$$

$$\text{subject to} \quad \sum_{j \in N} w_j x_j \leq c \tag{3}$$

$$\sum_{i \in N \setminus \{j\}} w_i y_{ij} \leq (c - w_j) x_j, \quad j \in N \tag{4}$$

$$0 \leq y_{ij} \leq x_j \leq 1, \quad i, j \in N, j \neq i \tag{5}$$

$$y_{ij} = y_{ji}, \quad i, j \in N, j > i \tag{6}$$

$$x_j, y_{ij} \in \{0, 1\}, \quad i, j \in N, j \neq i. \tag{7}$$

Constraints (5) and (6) allow a variable y_{ij} to be 1 only if x_j is 1. The reason for an explicit use of two distinct variables y_{ij} and y_{ji}, linked by equality constraints (6), will be clear in the following. Note that constraints

$$y_{ij} \leq x_i, \quad i, j \in N, j \neq i$$

need not be imposed explicitly, as they are implied by (5) and (6). Also, note that constraints

$$x_i + x_j \leq 1 + y_{ij}, \quad i, j \in N, j \neq i$$

forcing y_{ij} to be 1 when both x_i and x_j are 1, are unnecessary in the above ILP formulation, as all terms in the objective function are nonnegative. Even if these constraints could be used to tighten the *Linear Programming* (LP) relaxation obtained by removing (7), we do not consider them as they cannot be handled by our combinatorial algorithm for solving this LP relaxation.

Our main point is that, if equations (6) are removed, the resulting LP relaxation (2)–(5) can be solved in a very effective way. A *Continuous KP* (CKP) is a KP in which, for each item j, the constraint $x_j \in \{0, 1\}$ is replaced by $0 \leq x_j \leq 1$. Let p' and w' denote the profit and weight vectors of a CKP on n items, and c' the knapsack capacity. An optimal solution to the problem is easily obtained through a greedy algorithm due to Dantzig. Assume the items are sorted according to nonincreasing profit-over-weight ratios p'_j/w'_j, and let the *break item* be $b = \min\{h : \sum_{j=1}^{h} w'_j > c'\}$. Then an optimal solution of CKP is given by $x_j = 1$ for $j = 1, \ldots, b-1$ and $x_j = 0$ for $j = b+1, \ldots, n$, while the break variable takes the value $x_b = (c' - \sum_{j=1}^{b-1} w'_j)/w'_b$. A straightforward implementation of this algorithm runs in $O(n \log n)$ time, the bottleneck being item sorting. CKP was in fact shown to be solvable in $O(n)$ time through a median finding technique by Balas and Zemel [4].

Proposition 1 *An optimal solution $(\overline{x}, \overline{y})$ of (2)–(5) can be computed in $O(n^2)$ time by*

(i) solving the n CKPs associated with constraints (4), namely for $j \in N$:

$$\textit{maximize} \quad \overline{p}_j = \sum_{i \in N \setminus \{j\}} p_{ij} \pi_{ij}$$

$$
\begin{array}{lccccccccc}
\max & \boxed{\displaystyle\sum_{i\in N\setminus\{1\}} p_{i1}y_{i1}} & + & \boxed{\displaystyle\sum_{i\in N\setminus\{2\}} p_{i2}y_{i2}} & + & \boxed{\displaystyle\sum_{i\in N\setminus\{3\}} p_{i3}y_{i3}} & + \ldots + q_1x_1 & +q_2x_2 & +q_3x_3 & + \ldots \\
\text{s.t.} & \boxed{\displaystyle\sum_{i\in N\setminus\{1\}} w_iy_{i1}} & & & & & -(c-w_1)x_1 & & & \leq 0 \\
 & & & \boxed{\displaystyle\sum_{i\in N\setminus\{2\}} w_iy_{i2}} & & & & -(c-w_2)x_2 & & \leq 0 \\
 & & & & & \boxed{\displaystyle\sum_{i\in N\setminus\{3\}} w_iy_{i3}} & & & -(c-w_3)x_3 & \leq 0 \\
 & & & & & & w_1x_1 & +w_2x_2 & +w_3x_3 + \ldots & \leq c \\
 & 0 \leq y_{ij} \leq x_j \leq 1 \quad i,j \in N,\ i \neq j & & & & & & & &
\end{array}
$$

Figure 1: The relaxed problem (2)–(5) written in matrix form.

$$
\begin{aligned}
\textit{subject to} \quad & \sum_{i\in N\setminus\{j\}} w_i\pi_{ij} \leq (c-w_j) \\
& 0 \leq \pi_{ij} \leq 1, \quad i \in N\setminus\{j\},
\end{aligned} \tag{8}
$$

with optimal solution $\overline{\pi}_{ij}$, $i \in N\setminus\{j\}$;

(ii) solving the CKP associated with constraint (3) and profits $\overline{p}_j + q_j$, $j \in N$, *namely:*

$$
\begin{aligned}
\textit{maximize} \quad & \sum_{j\in N} (\overline{p}_j + q_j)x_j \\
\textit{subject to} \quad & \sum_{j\in N} w_jx_j \leq c \\
& 0 \leq x_j \leq 1, \quad j \in N,
\end{aligned} \tag{9}
$$

with optimal solution $\overline{x}_j$, $j \in N$;

(iii) defining $\overline{y}_{ij} := \overline{\pi}_{ij}\overline{x}_j$ *for* $i,j \in N, i \neq j$.

Proof. As illustrated in Figure 1, the relaxed problem has a special diagonal form. Namely, for each $j \in N$, variables y_{ij} ($i \in N\setminus\{j\}$), besides having a lower bound of 0 and a variable upper bound of x_j, appear only in constraint (4) associated with j, and in the objective function. Hence, if variable x_j is fixed to value $\overline{x}_j$ for all $j \in N$, the relaxed problem decomposes into n independent subproblems, one for each $j \in N$, of the form

$$
\begin{aligned}
\text{maximize} \quad & \sum_{i\in N\setminus\{j\}} p_{ij}y_{ij} \\
\text{subject to} \quad & \sum_{i\in N\setminus\{j\}} w_iy_{ij} \leq (c-w_j)\overline{x}_j \\
& 0 \leq y_{ij} \leq \overline{x}_j, \quad i \in N\setminus\{j\},
\end{aligned}
$$

which is clearly equivalent to the CKP (8), through the variable substitution $\pi_{ij} := y_{ij}/\overline{x}_j$ for $i \in N \setminus \{j\}$. (In fact, if $\overline{x}_j = 0$, then one has $\overline{y}_{ij} = 0$ for $i \in N \setminus \{j\}$.) Each subproblem yields the optimal value $\overline{y}_{ij}$ of variable y_{ij}, $i \in N \setminus \{j\}$.

Therefore, the contribution to the objective function by setting $x_j = \overline{x}_j$ is given by $(\overline{p}_j + q_j)\overline{x}_j$, where $\overline{p}_j$ is the optimal solution value of (8), independent of the values assigned to the other variables x_i, $i \in N \setminus \{j\}$. The determination of an optimal vector $\overline{x}$ then reduces to the CKP (9). □

By using the same arguments, it is easy to show that any optimal solution of the ILP (2)–(5) and (7) can be computed by solving n KPs analogous to (8), and then a KP analogous to (9). Of course, this approach yields better upper bounds, but our computational experience suggested working with the LP relaxation (2)–(5) as its solution is considerably faster (see the computational results of Section 3).

A relaxation very similar to that discussed above was also implicitly considered by Gallo, Hammer and Simeone [13]. These authors introduced the concept of *upper plane*, which is a linear function g satisfying $g(\overline{x}) \geq \sum_{j \in N} \sum_{i \in N} p_{ij} \overline{x}_i \overline{x}_j$ for any feasible solution $\overline{x}$ of (1). Clearly, an upper bound for QKP can be computed by optimizing g over the set of feasible solutions of (1), i.e., by solving a KP. As an alternative, one can solve the CKP associated with this KP. The actual upper planes proposed in [13] are of the form $\sum_{j \in N} \gamma_j x_j$, where, for $j \in N$, the following possible values of γ_j are considered:

(i) $\gamma_j := \sum_{i \in N} p_{ij}$;

(ii) $\gamma_j := \max\{\sum_{i \in N} p_{ij}\pi_i : \sum_{i \in N} \pi_i \leq d, \pi_i \in \{0,1\} \text{ for } i \in N\}$, where d is the maximum cardinality of a feasible QKP solution, i.e., assuming $w_1 \leq w_2 \leq \ldots \leq w_n$, $d := \max\{l \in N : \sum_{j=1}^{l} w_j \leq c\}$ — in this case γ_j is the sum of the d biggest profits among $\{p_{1j}, \ldots, p_{nj}\}$;

(iii) $\gamma_j := \max\{\sum_{i \in N} p_{ij}\pi_i : \sum_{i \in N} w_i\pi_i \leq c, 0 \leq \pi_i \leq 1 \text{ for } i \in N\}$;

(iv) $\gamma_j := \max\{\sum_{i \in N} p_{ij}\pi_i : \sum_{i \in N} w_i\pi_i \leq c, \pi_i \in \{0,1\} \text{ for } i \in N\}$.

Gallo, Hammer and Simeone experimentally showed that the upper plane corresponding to (iii) gives the best trade-off between tightness and computational effort.

It is immediate to see that coefficients (ii) to (iv) can be improved by forcing $\pi_j = 1$ in the computation of γ_j. In this case, the upper bounds computed by the Gallo-Hammer-Simeone approach coincide, respectively, with the optimal solution values of:

(i) (2),(3),(5), and (7) (or (2),(3),(5), if the continuous relaxation of the final KP is solved);

(ii) (2),(3),(5), and (7) (or (2),(3),(5), if the continuous relaxation of the final KP is solved) with the additional constraints $\sum_{i \in N \setminus \{j\}} y_{ij} \leq d - 1$ for $j \in N$;

(iii) (2)–(5), if the continuous relaxation of the final KP is solved;

(iv) (2)–(5) and (7).

Therefore, an upper bound computed by solving (2)–(5) is very similar to the one used by the most effective algorithm presented in [13]. While it is not clear how to improve this upper bound if it is presented within the upper plane framework, it is immediate to see how to tighten it by looking at the ILP formulation (2)–(7). Indeed, constraints (6), which are removed in the computation of this bound, can be relaxed in a Lagrangian way, as described below.

Lagrangian Relaxation

We introduce a matrix $\Lambda = (\lambda_{ij})$, where, for $i, j \in N, j > i$, λ_{ij} is the *Lagrangian multiplier* associated with the corresponding equation in (6) and, for notational convenience, $\lambda_{ji} := -\lambda_{ij}$. Accordingly, the Lagrangian modified objective function reads:

$$\text{maximize} \quad z(L(\text{QKP}, \Lambda)) = \sum_{j \in N} \sum_{i \in N \setminus \{j\}} \hat{p}_{ij} y_{ij} + \sum_{j \in N} q_j x_j, \tag{10}$$

where, for $i, j \in N, j \neq i$, $\hat{p}_{ij} := p_{ij} + \lambda_{ij}$ is the *Lagrangian profit* associated with variable y_{ij}.

The corresponding Lagrangian relaxed problem is given by (10) subject to (3)–(5) and (7). For a given Λ, the continuous relaxation of this problem (i.e., (10) subject to (3)–(5)) can be solved by the algorithm in Proposition 1. To this end, just observe that, in the solution of the n CKPs (8), if a modified profit happens to be nonpositve, the corresponding variable can be fixed at 0.

As our aim is determining a matrix Λ^* such that $z(L(\text{QKP}, \Lambda^*)) = \min_\Lambda z(L(\text{QKP}, \Lambda))$, we prove below that, without loss of generality, $\hat{p}_{ij} \geq 0$ for all $i, j \in N, j \neq i$. Indeed, for $i, j \in N, j \neq i$ one has $\hat{p}_{ij} + \hat{p}_{ji} = p_{ij} + p_{ji} = 2p_{ij}$, i.e., the determination of an optimal Λ corresponds to splitting each profit $2p_{ij}$ between the two objective function coefficients $\hat{p}_{ij}$ and $\hat{p}_{ji}$, so that the optimal solution of the relaxed problem is minimized. Hence,

Remark 1 *An optimal multiplier matrix Λ^* exists such that $\hat{p}_{ij} \geq 0$ for all $i, j \in N, j \neq i$.*

Proof. For a given Λ, suppose a pair $i, j \in N, j > i$ exists such that $\hat{p}_{ij} < 0$ (the case $\hat{p}_{ji} < 0$ is analogous). In this case, redefining $\lambda_{ij} := \lambda_{ij} - \hat{p}_{ij}$, and hence $\lambda_{ji} := \lambda_{ji} + \hat{p}_{ij}$, one gets $\hat{p}_{ij} = 0$, $\hat{p}_{ji} = 2p_{ij}$. Noting that we have $y_{ij} = 0$ in both the initial and the new solution, and that $\hat{p}_{ji}$ has been decreased by the above transformation, we conclude that the solution value of the Lagrangian problem associated with the new Λ is not greater than that of the initial one. □

For each Λ such that $\hat{p}_{ji} \geq 0$ for $i, j \in N, j \neq i$, the corresponding Lagrangian profit matrix $\hat{P}$ defines a QKP instance which is equivalent to the initial one, i.e., we have a *reformulation* of the original problem. The reformulation associated with the best upper bound obtained at the root node is the one used throughout our branch-and-bound algorithm, in which we

do not apply subgradient optimization for the bound computation at the branching nodes other than the root, but just solve the Lagrangian subproblem (10) subject to (3)–(5) and to the branching constraints, corresponding to this reformulation. The use of problem reformulations for the Quadratic Assignment Problem was proposed by a few authors, see the paper by Carraresi and Malucelli [10] for a unified analysis of the various approaches.

A well-known result in Lagrangian relaxation (see, e.g., Fisher [12]) states that the upper bound $z(L(\mathrm{QKP}, \Lambda^*))$, where Λ^* is an optimal multiplier matrix, coincides with the optimal value of the LP relaxation (2)–(6). Anyway, exact solution of this LP relaxation would be computationally very expensive due to the large number of variables and constraints involved (see also [6]). Our approach determines a near-optimal multiplier matrix Λ by a standard *subgradient optimization* procedure; see Held and Karp [15] and Held, Wolfe and Crowder [16]. The procedure generates a series $\Lambda^0, \Lambda^1, \Lambda^2, \ldots$ of matrices, where $\Lambda^0 := 0$ and, for $k \geq 0$, Λ^{k+1} is defined from Λ^k as follows. Let $(\overline{x}, \overline{y})$ denote an optimal solution of the Lagrangian relaxation associated with Λ^k. The corresponding subgradient vector is given by

$$\delta_{ij} := \overline{y}_{ij} - \overline{y}_{ji} \qquad i, j \in N, j > i \tag{11}$$

Using the technique proposed by Held and Karp, we compute the new multipliers by

$$\lambda_{ij}^{k+1} := \begin{cases} \lambda_{ij}^k & \text{if} \quad |\delta_{ij}| \leq \epsilon \\ \max(\lambda_{ij}^k - \gamma\delta_{ij}, -p_{ij}) & \text{if} \quad \delta_{ij} > \epsilon \\ \min(\lambda_{ij}^k - \gamma\delta_{ij}, p_{ij}) & \text{if} \quad \delta_{ij} < -\epsilon \end{cases} \tag{12}$$

for $i, j \in N, j > i$. Here, the step size γ is defined by

$$\gamma = \mu \frac{u - z^*}{\sum_{ij} \delta_{ij}^2} \tag{13}$$

where μ is a suitable parameter, while u and z^* are the values of the best upper bound and feasible QKP solution value found so far, respectively. In our implementation, the step size parameter μ is initially set to 1, and halved if the upper bound does not decrease within 20 consecutive iterations. The tolerance ϵ is set to 10^{-6}. The number of iterations is in each case limited by $200 + n$, since we experimentally observed that afterwards no substantial improvement occurs. The overall complexity of the upper bound computation is therefore $O(n^3)$.

2 The Branch-and-Bound Algorithm

Our branch-and-bound algorithm is based on the upper bounding procedure described in the previous section. At the root node of the branching tree, we apply subgradient optimization with an embedded heuristic procedure so as to define tight upper and lower bounds, as well as a convenient problem reformulation. Subgradient is followed by a reduction procedure in which we try to fix the value of some variables. If the reduction procedure fixes at least one variable, we apply subgradient optimization to the reduced

problem, followed by a new reduction. The process is iterated until no variable is fixed in the reduction.

The nodes of the branching tree other than the root are processed quite fastly, without any heuristic, reduction, or updating of the Lagrangian multipliers. We simply solve one Lagrangian relaxed subproblem (associated with the best multipliers found at the root node), in linear expected time, possibly updating the incumbent solution and applying branching.

We also considered a version of the branch-and-bound algorithm in which the Lagrangian multipliers are updated at each node, by applying subgradient optimization, starting from the best multipliers found at the father node. Although the upper bounds are slightly better, the overall computing time is much worse, at least for the instances in our test bed.

We next describe in detail each part of the branch-and-bound algorithm outlined above.

Heuristics

In order to derive a good initial solution, we implemented the heuristic devised by Billionnet and Calmels [6]. This algorithm first generates a greedy solution by initially setting $x_j = 1$ for $j \in N$, and then iteratively setting the value of a variable from 1 to 0, so as to achieve the smallest loss in the objective value, until a feasible solution is obtained. In the second step a sequence of iterations is performed in order to improve the solution by local exchanges. Let $S = \{j \in N : x_j = 1\}$ be the set of the items selected in the current solution. For each $j \in N \setminus S$, if $w_j + \sum_{\ell \in S} w_\ell \leq c$ set $I_j = \emptyset$ and let the quantity δ_j be the objective function increase when x_j is set to 1. Otherwise, let δ_j be the largest profit increase when setting $x_j = 1$ and $x_i = 0$ for some $i \in S$ such that $w_j - w_i + \sum_{\ell \in S} w_\ell \leq c$, and let $I_j = \{i\}$. Choosing k such that $\delta_k = \max_{j \in N \setminus S} \delta_j$, the heuristic algorithm terminates if $\delta_k \leq 0$, otherwise the current solution is set to $S \setminus I_k \cup \{k\}$ and another iteration is performed.

The above heuristic is applied as the first step of our algorithm, while at each second iteration of the subgradient optimization procedure we derive a heuristic solution as follows. The LP solution of (10) subject to (3)–(5) is rounded down, yielding an integer solution x. Starting from x the improvement part of the above algorithm is performed. The solutions obtained this way are typically substantially different from each other, even for slightly different Lagrangian profits, showing that the heuristic algorithm is worth applying often during the subgradient procedure. Overall, at the end of the subgradient procedure, we typically have a near-optimal incumbent solution z^* (in fact, optimal in most cases).

Reduction

The size of a QKP instance may be considerably reduced by using some reduction rules from the classical KP. Assume that we have an incumbent solution x^* of value z^*. Let u_i^1 be an upper bound on the QKP obtained by imposing the additional constraint $x_j = 1$. If $u_i^1 \leq z^*$ then we can fix x_j at 0. Similarly, if u_i^0 is an upper bound on the QKP obtained by imposing the additional constraint $x_j = 0$ and $u_i^0 \leq z^*$ we can fix x_j at 1.

We apply the reduction procedure at the end of the subgradient phase, deriving upper bounds u_j^1 and u_j^0 in $O(n^2)$ time for each j by solving the Lagrangian relaxed problem (10) subject to (3)–(5) associated with the best Λ, further constrained by imposing $x_j = 1$ and $x_j = 0$, respectively. If variable x_j is fixed at any value we remove the corresponding row and column. Moreover, if it is fixed at 1, we also increase diagonal entry q_i by $p_{ij} + p_{ji}$, for $i \in N \setminus \{j\}$, and decrease c by w_j.

Computational experiments showed that after at most 10 combined applications of the subgradient and reduction procedures no variable was fixed in the reduction.

Branching Scheme

In the following, N denotes the set of variables that were not fixed by the reduction procedure. Moreover, $\hat{P} = (\hat{p}_{ij})$ is the Lagrangian profit matrix associated with the best upper bound found by the subgradient procedure, $q = (q_j)$ is the diagonal profit vector modified according to the variables fixed at 1 by the reduction, and $\overline{p} = (\overline{p}_j)$ contains the optimal objective function values of CKP's (8) associated with matrix $\hat{P}$ rather than P.

Our branch-and-bound algorithm is based on a depth-first search, where the order in which variables are fixed by branching is determined in advance at the root node, allowing for a considerable speed-up of the computation at each node, as described in the following. We branch first on the variables which have a high probability of taking the value 1 in the optimal solution. To this aim, for each item i, $i \in N$, we compute the quantity

$$\gamma_i = q_i + \max \left\{ \sum_{j \in N \setminus \{i\}} \hat{p}_{ji} x_j : \sum_{j \in N \setminus \{i\}} w_j x_j \leq c - w_i,\ 0 \leq x_j \leq 1, j \in N \setminus \{i\} \right\} \qquad (14)$$

which represents an upper bound on the profit obtained by setting variable x_i to 1. Quantity γ_i is analogous to the profit $\overline{p}_i + q_i$ in the objective function of (9). Nevertheless, while this latter profit is associated with the i-th column of P, γ_i is associated with its i-th row. The use of the quantities γ_i is motivated by the fact that we need a criterion to distinguish between more and less "promising" items, and (near-)optimal Lagrangian profits $\overline{p}_i + q_i$ are not suited for this purpose since, as computational experience has shown, they tend to be similar to each other. (In practice, the initial profits $\overline{p}_i + q_i$ are "flattened" by the subgradient optimization procedure.) We reorder the variables according to nonincreasing values of γ_i, and systematically branch on the variable with the smallest index among the unfixed ones.

In order to speed up the search, we store the vector $\underline{w}$ of the minimum weights defined by

$$\underline{w}_i = \min_{j \geq i} w_i \text{ for } i \in N. \qquad (15)$$

Obviously, whenever the branching mechanism has fixed variables x_j at $\overline{x}_j$, $j = 1, \ldots, i-1$, so that $\sum_{j=1}^{i-1} w_j \overline{x}_j + \underline{w}_i > c$, we can backtrack, since no other variable x_j, $j \geq i$, can be set to one.

The branching scheme can easily be described in a recursive way. Assuming that variables $x_j,\ j = 1, \ldots, i-1$, have been fixed at $\overline{x}_j$, we have the profit and weight sums

$$\Pi = \sum_{j=1}^{i-1} \sum_{k=1}^{i-1} p_{jk} \overline{x}_j \overline{x}_k \qquad\qquad \Omega = \sum_{j=1}^{i-1} w_j \overline{x}_j. \tag{16}$$

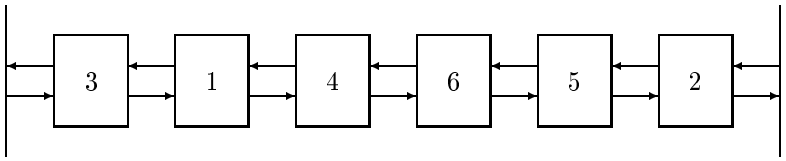
The next variable x_i can either be set to 1 or to 0. In the first case we update the diagonal elements for $j > i$ by setting $q_j \leftarrow q_j + p_{ji} + p_{ij}$. In the second case, no updating must be performed. In both cases, we recompute the break items associated with CKPs (8) as described in the next section.

As anticipated, subgradient optimization is applied only at the root node, whereas for the rest of the branch-and-bound algorithm we work on the QKP instance defined by the Lagrangian profit matrix $\hat{P}$. In particular, at each node, an upper bound is derived by solving problem (10) subject to (3)–(5), with c replaced by $c - \Omega$, on the unfixed items, and by adding Π to the optimal solution obtained. By using parametric techniques described in the next section, this upper bound can be computed in linear expected time. We backtrack if the upper bound u does not exceed the incumbent solution value z^*. This leads to the following recursive algorithm, which is initially called quadbranch(0,0,1), after the processing of the root node:

algorithm quadbranch(Π, Ω, i)
if $(\Pi > z^*)$ **then** $z^* \leftarrow \Pi$; $x^* \leftarrow x$;
if $(i \leq n)$ **and** $(\Omega + \underline{w}_i \leq c)$ **then**
  derive upper bound u;
  if $(u > z^*)$ **then**
    dequeue item i from each CKP given by (8), for $j = i+1, \ldots, n$;
    comment: branch on $x_i = 1$;
    find the new break item b in each problem (8), for $j = i+1, \ldots, n$;
    set $q_j \leftarrow q_j + p_{ji} + p_{ij}$ for $j = i+1, \ldots, n$;
    set $x_i \leftarrow 1$;
    call quadbranch$(\Pi + q_i, \Omega + w_i, i+1)$;
    set $q_j \leftarrow q_j - p_{ji} - p_{ij}$ for $j = i+1, \ldots, n$;
    comment: branch on $x_i = 0$;
    find the new break item b in each problem (8), for $j = i+1, \ldots, n$;
    set $x_i \leftarrow 0$;
    call quadbranch$(\Pi, \Omega, i+1)$;
    enqueue item i in each problem (8) for $j = i+1, \ldots, n$
  fi
fi.

Deriving Upper Bounds in Linear Expected Time

The upper bound computation is the most time-consuming operation at any node of the branching tree (corresponding to a recursive step of procedure quadbranch). It is therefore



i	s_i	$\hat{p}_{ij}$	w_i	π_i	σ_i
1	2	20	4	3	4
2	6	14	7	5	0
3	1	30	5	0	1
4	3	16	4	1	6
5	5	10	5	6	2
6	4	18	6	4	5

Figure 2: For each problem (8) we maintain a double-linked list of the active items ordered by nonincreasing profit/weight ratio, and an array of pointers to each element in the list.

extremely important to have a very efficient implementation of this part in order to limit the overall computing time.

A solution from scratch of the Lagrangian relaxed problem (10) subject to (3)–(5) (defined on the items not fixed by branching and on the profit matrix $\hat{P}$) would take $O(n^2)$ time, as described in Section 1, the bottleneck being the derivation of the profits $\overline{p}_j$ to be used in the objective function of the final CKP (9). Our approach solves the Lagrangian problem by using appropriate data structures to ensure an linear expected time complexity. The key observation is that all the $n-1$ items of each CKP (8) problem are present at the root node, while some of them are removed during the branching. This means that it is only necessary to order the items at the root node according to decreasing profit/weight ratios, and then use a double linked list to store the items which are still active, i.e., those whose variable has not been fixed by branching. Figure 2 shows the double linked list which is implemented by storing, for each item i, the sequence number s_i, the predecessor π_i and the successor σ_i, according to the sorting. The pointers to the list elements corresponding to each item are stored in an array, so that we can directly access each item in the list.

Let $K := \{k \in N : k \text{ unfixed}\}$, $L := \{k \in N : k \text{ fixed at } 1\}$ and $c' := c - w_j - \sum_{k \in L} w_k$, and consider problem (8) associated with an unfixed item j, where N and $c - w_j$ are replaced by K and c', respectively. For this problem we maintain the current break item position b, the sums $w' = \sum_{k \in K : s_k < b} w_k$ and $p' = \sum_{k \in K : s_k < b} \hat{p}_{kj}$. During the branching process we must ensure that the break item position b is defined by $\sum_{k \in K : s_k < b} w_k \leq c' < \sum_{k \in K : s_k \leq b} w_k$, since then the objective value of (8) is found as $\overline{p} = p' + (c' - w')\hat{p}_{tj}/w_t$ where t is such that $s_t = b$. Each iteration of the branch-and-bound algorithm may fix a variable x_i at 0 or 1, meaning that we must update b, p' and w' as follows:

1. $x_i = 1$, $s_i < b$: Both c' and w' are decreased by w_i while p' is decreased by $\hat{p}_{ij}$. The break item position b is unchanged.

2. $x_i = 1$, $s_i \geq b$: The capacity c' is decreased by w_i and we use linear search from b to the left in order to find the new value of b. Thus while $w' > c'$ repeatedly decrease p', w' by $\hat{p}_{tj}, w_t$, where t is such that $s_t = b - 1$, and decrease b by 1.

3. $x_i = 0$, $s_i > b$: All variables c', w', p' and b are unchanged.

4. $x_i = 0$, $s_i \leq b$: If $s_i < b$ then the sums p', w' are decreased by $\hat{p}_{ij}$ and w_i, otherwise b is incremented by 1. Linear search from b to the right is used to find the new value

of b. While $w' + w_t \leq c'$ increase p', w' by $\hat{p}_{tj}, w_t$, where t is such that $s_t = b$, and increment b by 1.

Since the item i may be accessed and dequeued/enqueued in constant time, both Steps 1 and 3 may be performed in constant time. Steps 2 and 4 involve a linear search which in the worst case may demand $O(n)$ time, but on average only demands a constant number of operations as shown below.

Assume that the weights are uniformly random distributed in an interval $[1, R]$ and that weight w_i of item i is independent of the corresponding profit/weight ratio $\hat{p}_{ij}/w_i$, so that the n weights in the ordered list may be seen as n independent random numbers in the interval $[1, R]$. Also assume that at each iteration of the branch-and-bound algorithm a randomly chosen variable i is fixed to $x_i = 0$ or $x_i = 1$. These assumptions are reasonable since every column knapsack problem of the form (8) basically leads to a different ordering of the items (and thus of the weights) and the branching process considers all different fixations of the variables.

Lemma 1 *A given random number a in $[1, R]$ is selected. On average, no more than 4 random numbers $b_1, b_2, \ldots$ should be drawn from the same interval $[1, R]$ before their sum exceeds a.*

Proof. Let $X_i \in \{0, 1\}$ be a stochastic variable whose value is 1 if the ith selected random number b_i is not smaller than $R/2$. Obviously $p(X_i{=}1) = \frac{1}{2}$. If we draw two numbers not smaller than $R/2$ then their weight sum is $R \geq a$. The expected number of coin flips X_i needed before $X_1 + X_2 + \ldots = 2$ is

$$E = \sum_{i=1}^{\infty} i \frac{i-1}{2^i} = \sum_{i=1}^{\infty} \frac{i^2}{2^i} - \sum_{i=1}^{\infty} \frac{i}{2^i} = 6 - 2 = 4 \tag{17}$$

since the probability of getting $X_i = 1$ with exactly one previous $X_j = 1$, $j < i$ is $(i-1)/2^i$. □

Tighter bounds than the above may be derived by a more thorough analysis, but the above is sufficient to prove the following.

Lemma 2 *The expected number of iterations in the forward/backward search of Steps 2 and 4 are bounded by a constant.*

Proof. For every item i fixed at $x_i = 1$ or $x_i = 0$ we must search backwards or forwards from the break item until the weight sum of the items passed is not smaller than w_i. With the assumption that each weight w_i is an independent random number in $[1, R]$ the statement follows from Lemma 1. □

The following proposition, derived from Lemma 2, expresses the main feature of our branch-and-bound algorithm.

Proposition 2 *Every node of the branching tree is processed in linear (in n) expected time.*

Proof. In a forward step of the branch-and-bound algorithm, from Lemma 2, each problem (8) is solved in constant expected time, and thus relaxation (10) subject to (3)–(5) is solved in linear expected time. Backtracking is performed in a similar way as forward steps, by enqueing an item. Finally, if the variables b, p', w', c' are stored as part of each branching node, no additional computation is needed. □

3 Computational Experiments

Our algorithm was implemented in ANSI C and run on a HP9000/735 workstation. We considered several classes of QKP instances which were presented in the literature. Gallo, Hammer and Simeone [13] solved some randomly generated instances, which form also the benchmark for the algorithms by Billionnet and Calmels [6] and Michelon and Veilleux [24]. Recently, Helmberg, Rendl and Weismantel [17] presented a compiler design problem which may be formulated as a QKP, and reported computational results only for some instances of this problem. We report computational results on both classes. Finally, we investigate whether our QKP code may be used effectively for solving Max Clique problems.

Randomly Generated Instances

The randomly generated instances by Gallo, Hammer and Simeone are constructed as follows. Let Δ be the *density* of the instance, i.e., the percentage of non-zero elements in the profit matrix P. Each weight w_j is randomly distributed in $[1, 50]$ while the profits $p_{ij} = p_{ji}$ are nonzero with probability Δ, and in this case randomly distributed in $[1, 100]$. Finally, the capacity c is randomly distributed in $[50, \sum_{j=1}^{n} w_j]$. (Notice that Gallo, Hammer and Simeone actually chose the capacities in $[1, \sum_{j=1}^{n} w_j]$ but later papers have increased the lower limit.)

In Tables 1 and 2 we consider instances with densities $\Delta = 25\%, 50\%, 75\%$ and 100% and with n up to 200. For each size n, the entries are average values out of 10 instances. For each density, we report the results up to the highest value of n (multiple of 20) such that all 10 instances were solved to optimality within a time limit of 50000 seconds for each instance.

Table 1 compares the performance of the proposed algorithm when three different upper bounds are applied. First, U_1 is the bound obtained by solving (2)–(5). This is the bound for which Gallo, Hammer and Simeone obtained their best solution times. In practice this bound is obtained by skipping subgradient optimization in our algorithm. The next bound U_2 is the bound proposed here. Finally, U_3 is a tighter version of U_2 obtained by solving the KPs (8) and (9) to integer optimality using the Lagrangian profits obtained at the end of subgradient optimization. In each case the bounds were used for the reduction phase as well as during the branch-and-bound enumeration.

Table 1: Comparison of different bounds on randomly-generated instances with different densities Δ. Tests run on a HP9000/735.

		U_1			U_2			U_3		
Δ	n	gap (%)	reduced	total time	gap (%)	reduced	total time	gap (%)	reduced	total time
	20	15.23	9	0.02	3.19	14	0.16	1.35	18	0.20
	40	26.69	7	9.26	2.64	20	0.92	2.43	22	1.30
25%	60	17.32	12	2.16	0.45	46	1.90	0.26	50	2.39
	80	14.80	6	916.81	1.02	46	4.66	0.58	55	37.54
	100	57.53	0	—	2.54	21	217.21	2.24	29	—
	120	26.63	18	—	0.44	70	20.08	0.40	72	—
	20	19.04	8	0.02	4.09	14	0.15	1.68	17	0.21
	40	25.74	3	0.31	3.08	20	0.77	2.27	24	2.00
	60	27.38	2	5.91	1.98	27	1.72	1.61	31	9.76
50%	80	14.77	8	93.25	0.64	49	4.32	0.34	56	13.90
	100	34.96	0	—	2.22	36	29.29	2.19	38	1798.88
	120	22.74	0	—	1.17	44	17.08	1.07	53	427.46
	140	38.06	0	—	1.23	70	285.43	1.19	72	—
	160	16.83	0	—	0.70	82	91.42	0.60	89	—
	20	8.14	11	0.01	4.21	13	0.14	2.83	17	0.18
	40	18.75	4	0.12	2.08	25	0.77	1.49	30	1.12
	60	16.51	12	0.92	1.04	44	1.92	0.48	54	2.42
	80	15.39	0	13.63	0.80	47	3.61	0.70	50	12.65
75%	100	13.80	6	282.95	1.94	52	6.93	0.87	60	19.90
	120	11.48	27	1114.49	0.65	85	8.01	0.54	96	18.44
	140	16.89	0	—	0.82	73	20.36	0.77	78	669.20
	160	16.19	0	—	0.73	73	29.58	0.66	77	302.05
	180	20.78	20	—	0.50	105	62.18	0.49	109	205.98
	200	13.32	7	—	0.49	130	41.10	0.46	135	284.97
	20	9.16	10	0.02	4.91	12	0.14	1.05	18	0.20
	40	10.35	0	0.11	1.68	24	0.65	1.02	29	1.08
	60	5.82	12	0.43	1.04	43	1.40	0.64	50	1.81
	80	10.80	6	1.50	0.57	64	2.85	0.29	71	4.10
100%	100	10.60	0	16.95	0.30	78	6.24	0.25	82	8.25
	120	11.52	0	211.16	0.56	79	11.36	0.51	82	24.47
	140	10.48	0	4023.33	0.32	103	24.47	0.24	122	31.62
	160	9.65	20	3869.69	0.33	128	19.35	0.30	134	28.97
	180	9.83	6	—	0.27	133	39.55	0.24	137	68.06
	200	10.31	10	—	0.54	115	248.37	0.52	117	373.70

During the branching process, bounds U_1 and U_2 were derived in linear expected time by using the algorithm presented in Section 2. Bound U_3 was derived using the `combo` algorithm by Martello, Pisinger and Toth [22] for solving the corresponding KPs. In the table we report the percentage gap between the given bound U_i and the optimal solution value z^* at the root node, and state how many variables were fixed at their optimal value during the reduction. Finally, the total average solution time, expressed in seconds, is given for the instances that could be solved within the time limit.

Only instances up to $n = 200$ have been considered, but this is sufficient to show that our algorithm based on bound U_2 gives the best overall performance. Using the bound by Gallo, Hammer and Simeone, one is only able to solve dense instances up to $n = 160$, despite the fast bounding procedure. Using the original technique from [13], where bounds are derived in $O(n^2)$ time at each node of the branching tree, one should not expect to solve instances with n much larger than 100. The exact knapsack bound U_3 leads to the tightest bounds at the root node, and it is able to reduce slightly more items. But the extra effort for obtaining the tighter bounds does not pay off when it comes to the overall

Table 2: Performances of the algorithm on randomly-generated instances with different densities Δ. Tests run on a HP9000/735.

Δ	n	root time	gap.upper bound (%)	gap.lower bound (%)	optimal solution	reduced variables	b&b nodes	total time
	20	0.16	3.19	0.00	2954	14	23	0.16
	40	0.91	2.64	0.00	9088	20	350	0.92
25	60	1.86	0.45	0.00	27381	46	119	1.90
	80	3.82	1.02	0.03	54726	46	11367	4.66
	100	10.20	2.54	0.03	43631	21	1605951	217.21
	120	18.29	0.44	0.00	92674	70	9191	20.08
	20	0.15	4.09	0.00	4159	14	23	0.15
	40	0.74	3.08	0.00	20740	20	787	0.77
	60	1.60	1.98	0.06	43993	27	1648	1.72
50	80	4.15	0.64	0.00	108969	49	1391	4.32
	100	8.94	2.22	0.01	91888	36	149888	29.29
	120	13.55	1.17	0.02	188761	44	28172	17.08
	140	27.27	1.23	0.02	235532	70	1354323	285.43
	160	40.58	0.70	0.00	378740	82	186119	91.42
	20	0.14	4.21	0.00	8380	13	24	0.14
	40	0.76	2.08	0.00	30914	25	127	0.77
	60	1.88	1.04	0.00	62326	44	194	1.92
	80	3.52	0.80	0.08	139790	47	315	3.61
	100	6.73	1.94	0.00	192668	52	1312	6.93
	120	7.78	0.65	0.00	223658	85	1548	8.01
	140	18.53	0.82	0.01	345174	73	13097	20.36
	160	32.91	0.73	0.05	557747	73	22770	29.58
75	180	72.72	0.50	0.00	391612	105	1612	62.18
	200	48.10	0.49	0.00	448943	130	2769	41.10
	220	145.72	0.74	0.02	965934	93	312734	161.73
	240	126.69	0.32	0.03	1265548	146	72534	118.95
	260	178.40	1.08	0.05	947582	70	192434	246.64
	280	137.58	0.20	0.00	1209289	217	97351	117.84
	300	305.17	0.34	0.05	1968164	166	328454	310.94
	320	303.94	0.47	0.01	1846898	166	130712925	1866.78
	340	488.76	0.25	0.03	2657714	204	294083	461.85
	360	683.12	0.92	0.02	2169773	144	8952166	7110.57
	20	0.14	4.91	0.00	9010	12	57	0.14
	40	0.65	1.68	0.00	51083	24	303	0.65
	60	1.39	1.04	0.00	114026	43	425	1.40
	80	2.80	0.57	0.00	143464	64	2167	2.85
	100	6.21	0.30	0.00	260186	78	656	6.24
	120	10.80	0.56	0.00	361040	79	31133	11.36
	140	23.59	0.32	0.00	585777	103	44208	24.47
	160	18.31	0.33	0.00	548418	128	46425	19.35
	180	36.52	0.27	0.17	683870	133	360290	39.55
100	200	49.31	0.54	0.00	887270	115	7882641	248.37
	220	55.36	0.19	0.05	1538179	158	17871673	797.40
	240	65.99	0.20	0.01	1274079	189	49128	68.39
	260	89.39	0.19	0.00	2342355	193	497389133	5366.11
	280	169.12	0.17	0.00	1739347	207	21608076	2975.73
	300	145.93	0.12	0.00	2546921	255	3605738	282.81
	320	228.53	0.23	0.00	2710140	211	240073849	2605.59
	340	229.88	0.17	0.00	2304562	256	87743885	10142.78
	360	262.51	0.15	0.00	3451536	298	79584918	1925.04
	380	365.08	0.17	0.00	2788526	278	500075295	17759.76
	400	420.92	0.10	0.00	4083908	323	413737707	8808.57

solution times.

In Table 2 we consider instances with up to 400 items solved by our algorithm, which uses bound U_2. Again, for each size n, the entries are average values of 10 instances. For each density, we report the results up to the highest value of n (multiple of 20) such that all 10 instances were solved to optimality within a time limit of 50000 seconds for each instance. The first entry gives the time, in seconds, used at the root node for deriving upper and lower bounds as well as reducing variables, while the next two columns give the percentage gap between the upper/lower bound and the optimal solution value at the root node. The average optimal solution value is given in the next column, followed by the number of reduced variables. Finally, we give the number of branch-and-bound nodes investigated, and the average solution time in seconds.

One can observe that the upper and lower bounds are generally very tight, making it possible to reduce a majority of the variables, on average more than 75%. The total preprocessing takes a couple of minutes for the largest instances. Despite this effective preprocessing, the final branch-and-bound phase demands some hours and a huge number of nodes for the largest instances, as many variables have to be fixed by branching before closing the gap, despite the latter is typically very small already at the root node. Apparently the algorithm works best for high-density instances since the upper bounds are generally tighter in these cases. The lower bounds are in all cases nearly optimal. We sometimes observed quite different behaviors on instances associated with the same Δ and n.

The solution times presented show a significant improvement with respect to previously published algorithms, in particular for instances with high density. The algorithm by Billionnet and Calmels [6] is only able to solve all the instances up to $n = 30$, using about 30 seconds on our machine. The algorithm by Michelon and Veilleux [24] is slightly better, being able to solve all instances up to $n = 40$ in about 20 seconds on our machine. According to Hammer and Rader [14], the largest instances solvable by the algorithm of Gallo, Hammer and Simeone [13] have size $n = 75$ and require about 600 seconds on our machine. Finally, the largest instances with density $\Delta = 100\%$ solved in Chaillou, Hansen and Mahieu [9] and Hammer and Rader [14] have size $n = 100$ and require about 1400 and 350 seconds on our machine, respectively. These two approaches have a better behaviour for low-density than for high-density instances. For example, for $\Delta = 25\%$, they require about 800 and 30 seconds on our machine for $n = 100$, respectively.

Compiler Design Instances

The 12 compiler design instances presented by Helmberg, Rendl and Weismantel [17] are considered in Table 3. The first seven columns are as in Table 2, while the last two columns give the solution times, in seconds, and the percentage gap between the upper and lower bounds of the best algorithm by Helmberg, Rendl and Weismantel [17], which is based on the combined use of semidefinite programming and (linear) cutting planes, and does not guarantee finding an optimal solution. The latter tests were run on a Sun Sparcstation 10 which is about four times slower than our machine. In all cases the Helmberg-Rendl-

Table 3: Results on compiler design instances.

n	c	root time	gap.upper bound (%)	gap.lower bound (%)	optimal solution	reduced variables	b&b nodes	total time	total time	final gap (%)
30	450	0.45	11.15	0.00	1580	10	15	0.45	24	0.00
	512	0.65	6.04	0.00	1802	25	1	0.65	1570	0.00
	600	0.39	0.00	0.00	2326	30	1	0.39	91	0.00
45	450	0.77	3.21	0.00	2840	36	17	0.77	823	0.00
	512	0.93	5.57	0.00	3154	25	83	0.93	1800	1.58
	600	0.78	0.28	0.00	3840	43	1	0.78	182	0.00
47	450	0.77	3.64	0.00	1732	41	1	0.77	1870	0.00
	512	1.01	1.73	0.00	1932	45	1	1.01	500	0.00
	600	1.88	13.91	0.00	2186	16	46	1.88	1800	4.09
61	450	1.30	0.97	0.00	26996	57	1	1.30	203	0.00
	512	1.24	1.69	0.00	29492	55	3	1.24	1800	0.02
	600	1.38	1.54	0.00	32552	52	29	1.38	1800	0.33

Weismantel algorithm found the optimal solution, but for some instances it was not able to prove optimality within the given time limit of 1800 seconds (for instance 7, the table in [17] reports that optimality was proved after 1870 seconds).

The table shows that our algorithm is considerably faster than the Helmberg-Rendl-Weismantel approach. Whereas our algorithm terminates within 1 second on average, their algorithm is not able to prove optimality of four instances within the time limit. It is however seen that the upper bounds computed through the use of semidefinite programming are tighter than our bounds. Observe that, even if we have to apply branching for optimally solving these instances, the time spent at the root node essentially coincides with the overall time.

Max Clique Instances

As mentioned in the introduction, the problem of finding a clique of size k in a graph G may be formulated as a QKP, and thus a Maximum Clique problem can easily be found by either binary or linear search among the possible values of k. We tested both approaches, in particular our implementation of linear search starts with $k = 1$ and at each iteration increases by one the value of k, continuing if a clique of size k is found by our algorithm and stopping when the non-existence of such a clique is proved. In practice, this implementation of linear search turns out to work much better than binary search, since finding a clique of size k is relatively easy when such a clique exists, while proving non-existence is relatively difficult. Thus the following tables refer to the linear search version.

We note that the upper bound obtained by solving (2)–(5) is trivial and weak for these instances: If the profits are defined as in the introduction, $v_1, \ldots, v_k$ denote the k vertices with largest degree in G, and $d(v)$ denotes the degree of vertex v, then the upper bound value is $\sum_{j=1}^{k} \min\{k-1, d(v_j)\}$. Therefore, if there are k or more vertices with degree at least $k-1$, the upper bound value is $k(k-1)$ and does not exclude the existence of a clique of size k. Anyway, Lagrangian relaxation yields tighter bounds, which are sufficient to solve to optimality clique instances of reasonable size. Actually, we assign profits $p_{ij} = p_{ji} = 100$ for edges $(i, j) \in E$, so that there is some freedom to modify the Lagrangian profits, as we work with integer values.

First, we consider Max Clique instances for random graphs in Table 4. These graphs are generated by specifying the number of nodes n and the edge density Δ, representing the probability of each possible edge to be present in the graph. For edge densities $\Delta = 25\%, 50\%, 75\%, 90\%$ and $n \leq 400$, we report the average solution times out of 10 instances, up to the highest value of n for which all 10 instances were solved to optimality within our time limit of 50000 seconds. Only relatively small instances can be solved for high densities within the time limit, while low density instances can be solved up to $n = 400$ in reasonable time. These results may appear surprising in view of Table 2, which shows that our approach works better for dense QKP instances, but clique instances defined on sparse graphs are known to be much easier than those defined on dense graphs.

Table 4: Results on Max Clique instances for randomly generated graphs.

$n\backslash\Delta$	25%	50%	75%	90%
20	0.15	0.14	0.16	0.17
40	0.59	0.74	1.34	4.04
60	1.63	2.63	14.94	176.73
80	3.37	7.76	156.87	9524.70
100	6.60	22.71	954.80	—
120	13.65	65.57	5627.91	—
140	23.34	154.62	—	—
160	34.76	394.43	—	—
180	64.20	971.39	—	—
200	114.98	1868.11	—	—
220	167.72	3312.77	—	—
240	244.02	7211.18	—	—
260	361.99	11957.87	—	—
280	516.76	20528.53	—	—
300	654.41	—	—	—
320	750.37	—	—	—
340	1211.63	—	—	—
360	1391.10	—	—	—
380	1905.89	—	—	—
400	2659.21	—	—	—

Next, Table 5 considers the Max Clique instances from DIMACS implementation challenge, see [19]. Only small-size instances with $n \leq 300$ were run. The entries give the instance name, the size of the graph (nodes and edges), the solution time, the size of the clique found by our algorithm, and the optimal solution value when our algorithm did not terminate within the time limit of 50000 seconds.

Our algorithm was not able to solve any of the instances by Sanchis (`San, SanR` derived from Vertex Cover), since our initial heuristic generally failed to find a tight lower bound. We can solve 13 out of the remaining 25 small instances. Thus on the DIMACS benchmark the performances of our QKP algorithm are comparable to that of cutting plane approaches for Max Clique [3], whereas it is not competitive with the state-of-the art algorithms for the problem [19].

Table 5: DIMACS clique benchmarks.

instance	nodes	edges	time	Clique size	
brock200_1	200	14834	>50000	20	(21)
brock200_2	200	9876	2209.79	12	
brock200_3	200	12048	8613.09	15	
brock200_4	200	13089	>50000	17	(17)
c-fat200-1	200	1534	28.35	12	
c-fat200-2	200	3235	45.28	24	
c-fat200-5	200	8473	204.77	58	
gen200_p0.9_44	200	17910	>50000	36	(44)
gen200_p0.9_55	200	17910	>50000	40	(55)
hamming6-2	64	1824	84.75	32	
hamming6-4	64	704	1.70	4	
hamming8-2	256	316	>50000	128	(128)
hamming8-4	256	20864	>50000	16	(16)
johnson16-2-4	120	5460	13700.17	8	
johnson8-2-4	28	210	0.22	4	
johnson8-4-4	70	1855	29.01	14	
keller4	171	9435	14098.00	11	
p_hat300-1	300	10933	804.75	8	
p_hat300-2	300	21928	>50000	21	(25)
p_hat300-3	300	33390	>50000	34	(36)
C125.9	125	6963	>50000	33	(34)
C250.9	250	27984	>50000	40	(44)
DSJC125.5	125	3891	79.31	10	
DSJC250.5	250	15668	8605.10	13	
MANN_a9	45	918	318.96	16	

4 Conclusions

Quadratic optimization problems are considered to be extremely difficult, and thus only small instances have been solved in the literature. Our work has demonstrated that it is possible to solve large size QKP instances to proven optimality within reasonable computing time.

A main contribution of this paper is an efficient procedure for computing a tight upper bound based on Lagrangian relaxation. Even if the upper bound is in principle weaker than the one computed by alternative approaches based on linear or semidefinite programming, the time required for its computation is some orders of magnitude smaller than that required by these alternative approaches. Furthermore, by applying subgradient optimization only at the root node, and using appropriate data structures, we compute upper bounds in linear expected time for each node during the branch-and-bound algorithm. This combination allows for an exact solution of instances about one order of magnitude larger with respect to previous existing methods.

The main conclusion which can be derived from our work is that, for the exact solution of the QKP instances in the literature, it is better to avoid computing tight upper bounds with sophisticated techniques, and to set up a branch-and-bound algorithm based on a rather fast combinatorial bounding procedure, which takes advantage of a convenient problem reformulation obtained from Lagrangian relaxation. This is a direction of research which should be investigated in the future for other combinatorial optimization problems.

It is also worth noting that our algorithm, without modifications, is capable of solving Max Clique problems of reasonable size. Even if we are not competitive with the state-of-

the-art exact algorithms for Max Clique (see [19]), we still perform almost as well as the cutting-plane approaches proposed so far for this problem (see [3]).

Acknowledgments

The first and third authors acknowledge Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST) and Consiglio Nazionale delle Ricerche (CNR), Italy, for the support of this project. The second author acknowledges the EC Network DIMANET for supporting the research by European Research Fellowship No. ERBCHRXCT-94 0429.

We are grateful to Federico Malucelli for helpful discussions, to Christoph Helmberg for sending us the compiler design instances, and to two anonymous referees for their helpful comments.

References

[1] W.P. Adams and H.D. Sherali (1986), "A Tight Linearization and an Algorithm for Zero-One Quadratic Programming Problems", *Management Science* **32**, 1274–1290.

[2] E. Balas, S. Ceria and G. Cornuéjols (1993), "A Lift-and-Project Cutting Plane Algorithm for Mixed 0-1 Programs", *Mathematical Programming* **58**, 295–324.

[3] E. Balas, S. Ceria, G. Cornuéjols and G. Pataki (1996), "Polyhedral Methods for the Maximum Clique Problem", in D.S. Johnson and M.A. Trick (eds.), *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS Press.

[4] E. Balas and E. Zemel (1980), "An Algorithm for Large Zero-One Knapsack Problems", *Operations Research*, **28**, 1130–1154.

[5] M. Bellare, O. Goldreich and M. Sudan (1995), "Free Bits, PCPs and Non-Approximability — Towards Tight Results", *Proceedings of the 36th FOCS Conference*, 422–431, IEEE Computer Society Press.

[6] A. Billionnet and F. Calmels (1996), "Linear Programming for the 0-1 Quadratic Knapsack Problem", *European Journal of Operational Research* **92**, 310–325.

[7] A. Billionnet, A. Faye and E. Soutif (1997), "A New Upper-Bound and an Exact Algorithm for the 0-1 Quadratic Knapsack Problem", presented at *ISMP'97*, Lausanne, EPFL, August 24-29, 1997

[8] K.M. Bretthauer, B. Shetty and S. Syam (1995), "A Branch-and-Bound Algorithm for Integer Quadratic Knapsack Problems", *ORSA Journal on Computing* **7**, 109–116.

[9] P. Chaillou, P. Hansen and Y. Mahieu (1986), "Best Network Flow Bounds for the Quadratic Knapsack Problem", *Lecture Notes in Mathematics* **1403**, 226–235.

[10] P. Carraresi and F. Malucelli (1994), "A Reformulation Scheme and New Lower Bounds for the QAP", in P.M. Pardalos and H. Wolkowicz (eds.), *Quadratic Assignment and Related Problems*, DIMACS series in Discrete Mathematics and Theoretical Computer Science, 147–160, AMS Press.

[11] G. Dijkhuizen and U. Faigle (1993), "A Cutting-Plane Approach to the Edge-Weighted Maximal Clique Problem", *European Journal of Operational Research* **69**, 121–130.

[12] M.L. Fisher (1981), "The Lagrangian Relaxation Method for Solving Integer Programming Problems", *Management Science* **27**, 1–18.

[13] G. Gallo, P.L. Hammer and B. Simeone (1980), "Quadratic Knapsack Problems", *Mathematical Programming* **12**, 132–149.

[14] P.L. Hammer and D.J. Rader, Jr. (1997), "Efficient Methods for Solving Quadratic 0-1 Knapsack Problems", *INFOR* **35**, 170–182.

[15] M. Held and R.M. Karp (1971), "The Traveling Salesman Problem and Minimum Spanning Trees: Part II", *Mathematical Programming* **1**, 6–25.

[16] M. Held, P. Wolfe and H.P. Crowder (1974), "Validation of Subgradient Optimization", *Mathematical Programming* **6**, 62–88.

[17] C. Helmberg, F. Rendl and R. Weismantel (1996), "Quadratic Knapsack Relaxations Using Cutting Planes and Semidefinite Programming", in W.H. Cunningham, S.T. McCormick and M. Queyranne (eds.), *Proceedings of the Fifth IPCO Conference*, Lecture Notes in Computer science **1084**, 175–189, Springer Verlag.

[18] E.L. Johnson, A. Mehrotra and G.L. Nemhauser (1993), "Min-Cut Clustering", *Mathematical Programming* **62**, 133–152.

[19] D.S. Johnson and M.A. Trick (eds.) (1996), *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS Press.

[20] L. Lovász and A. Schrijver (1991), "Cones of Matrices and Set-Functions and 0-1 Optimization", *SIAM Journal on Optimization* **1**, 166–190.

[21] S. Martello and P. Toth (1990), *Knapsack Problems: Algorithms and Computer Implementations,* Wiley, Chichester, England.

[22] S. Martello, D. Pisinger and P. Toth (1997), "Dynamic Programming and Tight Bounds for the 0-1 Knapsack Problem", Working Paper, DIKU report 97/11, University of Copenhagen, to appear in *Management Science*.

[23] P. Michelon and N. Maculan (1993), "Lagrangean Methods for 0-1 Quadratic Programming", *Discrete Applied Mathematics* **42**, 257–269.

[24] P. Michelon and L. Veilleux (1996), "Lagrangean Methods for the 0-1 Quadratic Knapsack Problem", *European Journal of Operational Research* **92**, 326–341.

[25] K. Park, K. Lee, S. Park (1996), "An Extended Formulation Approach to the Edge-Weighted Maximal Clique Problem", *European Journal of Operational Research* **95**, 671–682.

[26] J. Rhys (1970), "A Selection Problem of Shared Fixed Costs and Network Flows", *Management Science* **17**, 200–207.

[27] C. Witzgall (1975), "Matematical Methods of Site Selection for Electronic Message Systems (EMS)", *NBS internal report*.