INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Solution of Large Quadratic Knapsack Problems Through Aggressive Reduction

W. David Pisinger, Anders Bo Rasmussen, Rune Sandvik,

To cite this article:
W. David Pisinger, Anders Bo Rasmussen, Rune Sandvik, (2007) Solution of Large Quadratic Knapsack Problems Through Aggressive Reduction. INFORMS Journal on Computing 19(2):280-290. https://doi.org/10.1287/ijoc.1050.0172

INFORMS Journal on Computing
Vol. 19, No. 2, Spring 2007, pp. 280–290
ISSN 1091-9856 | EISSN 1526-5528 | 07 | 1902 | 0280

DOI 10.1287/ijoc.1050.0172

Solution of Large Quadratic Knapsack Problems Through Aggressive Reduction

W. David Pisinger, Anders Bo Rasmussen, Rune Sandvik
DIKU, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark
{pisinger@diku.dk, fuzz@fuzz.dk, mail@runesandvik.dk}

The *quadratic knapsack problem* (QKP) calls for maximizing a quadratic objective function subject to a knapsack constraint. All coefficients are assumed to be nonnegative and all decision variables are binary. A new exact algorithm is presented, which makes use of *aggressive reduction* techniques to decrease the size of the instance to a manageable size. A cascade of upper bounds is used for the reduction, including an improved version of the Caprara-Pisinger-Toth bound based on upper planes and reformulation, and the Billionnet-Faye-Soutif bound based on Lagrangian decomposition. Generalized reduction techniques based on implicit enumeration are used to fix variables at their optimal values. In order to obtain lower bounds of high quality for the reduction, a core problem is solved, defined on a subset of variables. The core is chosen by merging numerous heuristic solutions found during the subgradient-optimization phase. The upper and lower bounding phases are repeated several times, each time improving the subgradient method used for finding the Lagrangian multipliers associated with the upper bounds. Having reduced the instance to a (hopefully) reasonable size, a branch and bound algorithm based on the Caprara-Pisinger-Toth framework is applied. Computational experiments are presented showing that several instances with up to 1,500 binary variables can be reduced to fewer than 100 variables. The remaining set of variables are easily handled through the exact branch and bound algorithm. In comparison to previous algorithms the framework does not only solve larger instances, but the algorithm also works well for instances with smaller densities of the profit matrix, which appear frequently when modeling various graph problems as quadratic knapsack problems.

Key words: programming, quadratic; programming, integer, algorithms, branch and bound; analysis of algorithms: computational complexity; programming, integer, algorithms, relaxation
History: Accepted by William J. Cook, Area Editor for Design and Analysis of Algorithms; received October 2003; revised September 2005; accepted October 2005.

1. Introduction

The binary *quadratic knapsack problem* (QKP) is defined as follows: Assume that a set $N = \{1, \ldots, n\}$ of items is given where item j has a positive integer weight w_j. In addition we are given an $n \times n$ nonnegative integer matrix $P = \{p_{ij}\}$, where $p_{ij} + p_{ji}$ is the profit achieved by selecting two different items i and j. Furthermore, profit p_{ii} is achieved if item i is chosen.

The QKP calls for selecting an item subset whose overall weight does not exceed a given knapsack capacity c, so as to maximize the overall profit. By introducing binary variables x_j to indicate whether item j is selected, the problem may be formulated:

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

Without loss of generality we assume that $\max_{j \in N} w_j \le c < \sum_{j \in N} w_j$. If the first inequality is violated, i.e., if $w_j > c$ for some j, we may fix the decision variable x_j to 0. If the second inequality is violated a trivial solution exists with all items chosen. Negative weights can be handled by flipping the variable x_j to $1 - x_j$ for each item with $w_j < 0$.

We may assume that the profit matrix is symmetric, i.e., $p_{ij} = p_{ji}$ for all $j > i$. If $p_{ij} \ge 0$ for all coefficients $i \ne j$ the QKP is denoted the *super-modular knapsack problem*. In the sequel we will rely on the stronger assumption that all coefficients $p_{ij} \ge 0$, $i, j \in N$, which is not made without loss of generality. QKP in all the above-mentioned forms is $\mathcal{NP}$-hard in the strong sense, which can be seen by reduction from the clique problem.

As one might expect, due to its generality, QKP has a wide spectrum of applications in, e.g., telecommunications (Witzgall 1975), location problems (Rhys 1970), compiler design (Johnson et al. 1993, Helmberg et al. 1996), and VLSI design (Ferreira et al. 1996). Moreover numerous graph problems can be formulated as QKP, e.g. the dense subgraph problem, the weighted maximum b-clique problem (Park et al. 1996), and QKP appears when solving the graph-partitioning problem (Johnson et al. 1993). Finally, the

QKP appears as a pricing problem in several network-design problems when using a column generation framework (Thomadsen and Larsen 2007).

Numerous upper bounds have been presented for the QKP. A family of upper bounds based on upper planes was presented by Gallo et al. (1980), while a bound based on Lagrangian relaxation of the capacity constraint was presented by Chaillou et al. (1989). Caprara et al. (1999) presented upper bounds based on Lagrangian relaxation and reformulation. Billionnet and Calmels (1996) presented a bound based on linearization. Bounds based on Lagrangian decomposition have been presented by Michelon and Veilleux (1996) and Billionnet et al. (1999). For a recent survey on knapsack problems and also the corresponding upper bounds see Kellerer et al. (2004). A comprehensive experimental comparison of the above bounds is presented in Rasmussen and Sandvik (2002).

Several reduction methods, used for fixing some of the decision variables at their optimal value, have been presented for the QKP, see, e.g., Hammer and Rader (1997) and Caprara et al. (1999). Recent advances in the solution of difficult $\mathcal{NP}$-hard optimization problems have shown the importance of good reduction techniques, see, e.g., the seminal work of Polzin and Vahdati (2002) for the Steiner problem. Following this direction of research, we present an exact algorithm for the QKP, where reduction plays a key role in the solution process.

We introduce the term *aggressive reduction* to denote preprocessing techniques with the following properties:

1. The reduction does not run in polynomial time.
2. A cascade of different upper and lower bounds is used for fixing variables.
3. Implicit-enumeration techniques are used in the reduction.

Although property 1 may not seem very attractive, it gives a good characterization of aggressive reduction. Most textbooks define "normal" reduction techniques as a set of cheap (i.e., polynomial) preprocessing steps, which can be used to fix some of the variables at their optimal value. In aggressive reduction, the preprocessing steps *are* the main solution approach, and hence the time complexity need not be polynomially bounded. Property 2 emphasizes the need for tight upper and lower bounds in a reduction algorithm. In aggressive reduction, numerous bounds are used. If one bound fails in reducing a variable, the algorithm should not give up, but instead try to use other bounds for the reduction. Property 3 generalizes the ordinary dichothomic reduction for binary variables, to a general framework where one may enumerate up to a given depth in the search tree when trying to reduce a variable.

We present an exact algorithm based on aggressive reduction followed by branch and bound. The reduction algorithm is an iterative process that is repeated as long as at least one variable was reduced. Two different upper bounds are used for the reduction: the upper bound U^2_{CPT} by Caprara et al. (1999) based on upper planes, and U^2_{BFS} by Billionnet et al. (1999) based on Lagrangian decomposition.

As observed in the computational study by Rasmussen and Sandvik (2002), the upper bound U^2_{CPT} has low computation time and good quality for instances with high density of the profit matrix. The slower upper bound U^2_{BFS} returns on average tighter upper bounds than does U^2_{CPT}, especially on low-density instances. We alternate between using U^2_{CPT} and U^2_{BFS}, using the fast bound U^2_{CPT} to reduce the instance to a manageable size, before employing the slower bound U^2_{BFS}. An additional motivation for alternating between the bounds is that U^2_{BFS} does not dominate U^2_{CPT} as shown in Rasmussen and Sandvik (2002). To strengthen further the U^2_{CPT}, we solve all associated knapsack subproblems to integer optimality as opposed to the original bound, where only the LP relaxation of the subproblems was solved.

Several new approaches are used to determine tighter lower bounds. This includes local search techniques and the solution of a core problem based on a merging of suboptimal solutions.

Even though we try to use the best available upper and lower bounds from the literature, this may not be sufficient to reduce the problem. In this case our use of enumerative reduction may enhance the reduction. The traditional way of reducing binary decision variables is to branch on the variable at the root node of an enumerative tree, and see if one of the child nodes can be fathomed through any bounding methods. If one of the child nodes is fathomed, we may fix the variable to the value of the opposite child node. Enumerative reduction is basically a generalization of this approach, where we enumerate all nodes up to a given depth. If bounding methods are able to fathom all nodes in the left subtree, we may fix the variable to the value associated with the right child node (and vice-versa).

The reduction algorithm terminates when no more variables can be fixed by any variant of the two bounds, and no improvements of the bounds can be done through subgradient optimization. In this case, we switch to a branch and bound algorithm proposed by Caprara et al. (1999). The algorithm first finds a reformulation of the profit matrix P such that the objective function $\sum_{i\in N}\sum_{j\in N} p_{ij}x_ix_j$ is unchanged for any integer solution, but such that upper bounds based on so-called upper planes give tighter bounds for the reformulated problem. These bounds are then used in the branch and bound algorithm. We improved the latter in two respects: the algorithm

makes use of a new branching order, and we are able to find a better reformulation of the profit matrix P due to improved methods for finding the associated Lagrangian multipliers.

In Section 2 we will present a number of upper bounds used in the reduction of the problem and also for pruning the branch and bound tree. Despite the good quality of the upper bounds, they are of little use if not matched by correspondingly good lower bounds. Hence Section 3 presents several heuristics used for obtaining a lower bound of high quality. Section 4 describes how the aggressive reduction algorithm is used to fix most possible variables at their optimal values, before proceeding to the branch and bound part, which is described in Section 5. Finally, Section 6 presents a number of computational experiments with the developed algorithm solving instances with up to 1,500 variables. The paper is concluded in Section 7 with a discussion of the obtained results and a summary of the theoretical ideas.

2. Upper Bounds

The choice of upper-bounding procedures to be used is based on a tradeoff between the tightness of the bound obtained and the time required for its computation. The good tradeoff between computation time and quality by the fast bound suggested by Caprara et al. (1999) makes it well suited for the branch and bound algorithm. In order to reduce the instance as much as possible before proceeding to the branch and bound phase, we also apply the slower but tighter bound by Billionnet et al. (1999) in the reduction phase.

2.1. Caprara, Pisinger, Toth

The bound by Caprara et al. (1999) can be described within the framework of upper planes as follows. The objective function can be rewritten as

$$\sum_{i\in N}\sum_{j\in N} p_{ij}x_ix_j = \sum_{j\in N}\left(p_{jj} + \sum_{i\in N\setminus\{j\}} p_{ij}x_i\right)x_j.$$

Noting that the expression inside the parenthesis cannot exceed

$$\pi_j := p_{jj} + \max\left\{\sum_{i\in N\setminus\{j\}} p_{ij}x_i : \sum_{i\in N\setminus\{j\}} w_ix_i \le (c-w_j),\ x_i\in\{0,1\} \text{ for } i\in N\setminus\{j\}\right\}, \quad (2)$$

then an upper bound U^1_{CPT} is derived as the optimal solution value to

$$\begin{aligned} \text{maximize} \quad & \sum_{j\in N}\pi_jx_j \\ \text{subject to} \quad & \sum_{j\in N} w_jx_j \le c \\ & x_j\in\{0,1\}, \quad j\in N. \end{aligned} \quad (3)$$

If we relax the integrality constraints in the above subproblems we obtain the bound U^{1*}_{CPT}, which can be derived in $O(n^2)$ time by solving the n LP relaxed knapsack problems (2), and one LP relaxed knapsack problem (3).

Caprara et al. (1999) further strengthened the bound by noting that the objective function can be reformulated as

$$\sum_{i\in N}\sum_{j\in N} p_{ij}x_ix_j = \sum_{i\in N}\sum_{j\in N}(p_{ij}+\lambda_{ij})x_ix_j \quad (4)$$

for any skew-symmetric matrix Λ. Let QKP(Λ) denote the problem

$$\begin{aligned} \text{maximize} \quad & \sum_{i\in N}\sum_{j\in N}(p_{ij}+\lambda_{ij})x_ix_j \\ \text{subject to} \quad & \sum_{j\in N} w_jx_j \le c \\ & x_j\in\{0,1\}, \quad j\in N, \end{aligned} \quad (5)$$

and $U^{1*}_{\text{CPT}}(\Lambda)$ the corresponding upper bound obtained by solving the n continuous KP on the form (2) and the continuous KP on the form (3). In order to obtain the tightest bound we solve the Lagrangian dual problem

$$U^2_{\text{CPT}} = \min_{\{\lambda_{ij}:\lambda_{ij}=-\lambda_{ji}\}} U^{1*}_{\text{CPT}}(\Lambda). \quad (6)$$

The latter problem may be solved through subgradient optimization leading to the bound $\widehat{U}^2_{\text{CPT}}$ for some near-optimal matrix Λ of Lagrangian multipliers. Obviously $U^2_{\text{CPT}} \le \widehat{U}^2_{\text{CPT}} \le U^{1*}_{\text{CPT}}(0)$.

To further tighten the bound, we use a variant of $\widehat{U}^2_{\text{CPT}}$ where (2) and (3) are solved as integer knapsack problems. Solving integer KP instead of a continuous KP leads to slightly longer computation time of the bound in practice, but the total running time of the algorithm decreases due to the better quality of the bound.

2.2. Billionnet, Faye, Soutif

The bound by Billionnet et al. (1999) is based on a partitioning of N into m disjoint classes $\{I_1,\dots,I_m\}$ satisfying $\bigcup_{k=1}^m I_k = N$. The main idea in the bound by Billionnet et al. (1999) is to use Lagrangian decomposition to split the problem into m independent subproblems. Each subproblem is solved by enumerating all decision variables in class I_k. For a fixed solution vector x_{I_k} the subproblem is an ordinary linear 0-1 knapsack problem that may be solved in $O(n)$ time.

In order to partition the problem we notice that the objective function of QKP may be written as

$$\sum_{i\in N}\sum_{j\in N} p_{ij}x_ix_j = \sum_{k=1}^{m}\left(\sum_{i\in I_k}\sum_{j\in I_k} p_{ij}x_ix_j + \sum_{i\in I_k}\sum_{j\in N\setminus I_k} p_{ij}x_ix_j\right).$$

Using Lagrangian decomposition we introduce binary copy variables $y_j^k = x_j$ for $j \in N \backslash I_k$ and add k redundant capacity constraints. Moreover we add copy constraints concerning the quadratic terms $x_i x_j = x_i y_j^k$, getting the following formulation. The function set(i) returns the set index of the class to which item i belongs, i.e., the index k for which $i \in I_k$. Since the sets I_k are disjoint, set(i) is well defined. Hence, we get the formulation

$$\begin{aligned}
\text{maximize} \quad & \sum_{k=1}^{m}\left(\sum_{i\in I_k}\sum_{j\in I_k} p_{ij}x_i x_j + \sum_{i\in I_k}\sum_{j\in N\backslash I_k} p_{ij}x_i y_j^k\right) \\
\text{subject to} \quad & x_j = y_j^k \quad k = 1,\ldots,m,\ j \in N\backslash I_k \\
& x_i y_j^{\text{set}(i)} = x_j y_i^{\text{set}(j)} \\
& \qquad i \in N,\ j \in N,\ \text{set}(i) \neq \text{set}(j) \qquad (7)\\
& \sum_{i\in I_k} w_i x_i + \sum_{j\in N\backslash I_k} w_j y_j^k \le c \quad k = 1,\ldots,m \\
& x_i \in \{0,1\} \quad i \in I_k,\ k = 1,\ldots,m \\
& y_j^k \in \{0,1\} \quad k = 1,\ldots,m,\ j \in N\backslash I_k.
\end{aligned}$$

Lagrangian relaxing the two first constraints using multipliers $\Lambda = \{\lambda_j^k\}$, $k = 1,\ldots,m$, $j \in N\backslash I_k$ and respectively $M = \{\mu_{ij}\}$, $i \in N$, $j \in N$, $\text{set}(i) \neq \text{set}(j)$ we can decompose (7) into m independent problems (see Billionnet et al. (1999) or Kellerer et al. (2004) for details). The m problems $L_k(I_k, \Lambda, M)$, $k = 1,\ldots,m$ are of the form

$$\begin{aligned}
\text{maximize} \quad & \sum_{i\in I_k}\sum_{j\in I_k} p_{ij}x_i x_j + \sum_{i\in I_k}\left(\sum_{h\neq k}\lambda_i^h\right)x_i \\
& + \sum_{j\in N\backslash I_k}\left(\sum_{i\in I_k} p_{ij}x_i\right)y_j^k - \sum_{j\in N\backslash I_k}\lambda_j^k y_j^k \\
& + \sum_{j\in N\backslash I_k}\left(\sum_{i\in I_k}(\mu_{ij}-\mu_{ji})x_i\right)y_j^k \qquad (8)\\
\text{subject to} \quad & \sum_{i\in I_k} w_i x_i + \sum_{j\in N\backslash I_k} w_j y_j^k \le c \\
& x_i \in \{0,1\} \quad i \in I_k,\ k = 1,\ldots,m \\
& y_j^k \in \{0,1\} \quad j \in N\backslash I_k.
\end{aligned}$$

Assuming that the sets I_k are small we may enumerate all solutions of I_k in problem $L_k(I_k, \Lambda, M)$. For a fixed value of x_i, $i \in I_k$, the problem can be recognized as a 0-1 knapsack problem defined in the variables y_j^k by setting $\tilde{p}_j = \sum_{i\in I_k} p_{ij}x_i - \lambda_j^k + \sum_{i\in I_k}(\mu_{ij}-\mu_{ji})x_i$ and hence getting $L_k(I_k, \Lambda, M)$ in the form

$$\begin{aligned}
\text{maximize} \quad & d_j + \sum_{j\in N\backslash I_k}\tilde{p}_j y_j^k \\
\text{subject to} \quad & \sum_{j\in N\backslash I_k} w_j y_j^k \le c - \sum_{i\in I_k} w_i x_i \qquad (9)\\
& y_j^k \in \{0,1\}, \quad j \in N.
\end{aligned}$$

where $d_j = \sum_{i\in I_k}\sum_{j\in I_k} p_{ij}x_i x_j + \sum_{i\in I_k}(\sum_{h\neq k}\lambda_i^h)x_i$ is a constant. The objective function may hence be written $U_{\text{BFS}}^1(\Lambda, M) = \sum_{k=1}^m L_k(I_k, \Lambda, M)$. The tightest bound U_{BFS}^2 is now found as a solution to the Lagrangian dual problem

$$U_{\text{BFS}}^2 = \min_{\Lambda, M} U_{\text{BFS}}^1(\Lambda, M). \qquad (10)$$

A suboptimal choice of Lagrangian multipliers Λ and M can be found through subgradient optimization, leading to the bound $\hat{U}_{\text{BFS}}^2$ with $U_{\text{BFS}}^2 \le \hat{U}_{\text{BFS}}^2 \le U_{\text{BFS}}^1(0,0)$.

In order to make the computation of $U_{\text{BFS}}^1(\Lambda, M)$ faster we solve the continuous relaxation of (9), which can be done in $O(n)$ time.

The time complexity of the bound in this case for given values of Λ, M is derived as follows. If we assume that the set N was partitioned into m sets I_k of equal size n/m then the time consumption for solving problem $L_k(I_k, \Lambda, M)$ for a given choice of λ, μ is $O(2^{n/m}n)$. As we have to solve m subproblems the total time consumption is $O(2^{n/m}mn)$. The time complexity should be multiplied by the number of iterations of the Lagrangian multipliers in the bound $\hat{U}_{\text{BFS}}^2$. In their computational experiments, Billionnet et al. (1999) used relatively small sets of size $n/m = 5$ to keep the computational times at a reasonable level.

We note that the quality of the bound depends not only on the size of the groups and the quality of the Lagrangian multipliers, but also on how elements are assigned to groups. It is, however, not clear which assignment of elements to groups results in the tightest bound. The assignment of elements to groups is therefore done at random.

In the reduction phase of the algorithm it is required to add constraints of the type $x_i = v$ where $v \in \{0,1\}$ to the bound. Suppose we add the constraint $x_i = v$ to the reformulation (7). The constraint $x_j = y_j^k$ then implies $y_j^k = v$ for $k = 1,\ldots,m$ and $j = N\backslash\{I_k\}$. We can therefore enforce the additional constraint by fixing $y_i^k = v$ and $x_i = v$ in (9).

3. Lower Bounds

In the reduction phase of the algorithm several heuristics are used to search for good initial solutions.

In order to derive an initial solution, we implemented the heuristic devised by Billionnet and Calmels (1996). 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 to improve the solution by local exchanges. Let $S = \{j \in N\colon x_j = 1\}$ be the set of the items selected in the current solution. For each $j \in N\backslash S$, if $w_j + \sum_{l\in S} w_l \le c$

set $I_j = \varnothing$ and let 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 $w_j - w_i + \sum_{l \in S} w_l \le c$, and let $I_j = \{i\}$. Choosing k such that $\delta_k = \max_{j \in N \setminus S} \delta_j$, the heuristic algorithm terminates if $\delta_k \le 0$; 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. In every second step in the subgradient algorithm in the bound suggested by Caprara et al. (1999) the local-exchanges part of the above algorithm is performed. The initial solution for the local-exchange heuristic is given by the solution of the integer version of the knapsack problem (3).

To improve on the lower bound found by this heuristic we suggest a simple method based on *tabu search*. We use the same neighborhood as described above. A move (i, j) in the neighborhood is defined by removing an item i from the solution and adding an item j. Clearly a move (i, j) is legal only if $w_j - w_i + \sum_{l \in S} w_l \le c$. Let δ_{ij} be the change in the objective function by applying the move (i, j). We maintain a *tabu list* T of moves temporarily disallowed. In each iteration the legal move is located that gives the greatest increase in the objective function without being temporarily disallowed. That is, we choose (h, k) such that

$$\delta_{hk} = \max_{\{i \in N,\, j \in N \setminus S \mid w_j - w_i + \sum_{l \in S} w_l \le c \text{ and } (i,j) \notin T\}} \delta_{ij}$$

The current solution is set to $S \setminus \{h\} \cup \{k\}$ and (h, k) is added to the list of temporarily disallowed moves T. A move is removed from T after 500 iterations and the algorithm is terminated after completing 5,000 iterations.

Both the local-exchange heuristic and the tabu-search-based heuristic search solutions only with the same cardinality as the initial solution given to the heuristic. We therefore restart these heuristics with different initial solutions. The local-exchange heuristic is restarted every second iteration of the subgradient algorithm, while the tabu-search heuristic runs only in the first iteration.

As a third heuristic, we suggest an algorithm that solves a *core problem*. The core of the problem is defined by merging several high-quality solutions. Those variables that have the same solution values in nearly all heuristic solutions are fixed, while the remaining variables define a core of the problem, which is solved to optimality using the same algorithm. This approach corresponds to the tour-merging approach presented by Cook and Seymour (2003), and also to the ordinary core problem for a knapsack problem as presented by Balas and Zemel (1980).

To be more specific, the core of the QKP is defined as follows. During calculation of the upper planes in the subgradient procedure, we obtain some statistical information from the corresponding knapsack problems solved. Let f_j^1 be the number of times a variable x_j was set to 1 in the KP, and similarly f_j^0 is the number of times it was set to 0. Obviously, if $f_j^1 \gg f_j^0$ we may assume that $x_j = 1$ in an optimal solution, while if $f_j^0 \gg f_j^1$ we may assume that $x_j = 0$, and hence we tentatively fix the variables at the corresponding value. To define the core C of the problem we sort the items according to nondecreasing values of $|f_j^1 - f_j^0|$ and choose the first $n/4$ items. The remaining variables are temporarily fixed to $x_j = 1$ if $f_j^1 > f_j^0$ and $x_j = 0$ otherwise. The QKP defined on the core C is solved to optimality if this can be accomplished in under 2,000,000 iterations in the branch and bound algorithm. If not, the best solution found so far is returned. This heuristic succeeds in finding a better lower bound than previously-suggested heuristics in some of the very hard instances. The core algorithm is computationally quite expensive, and it is therefore used only when the algorithm has failed in reducing the instance to a manageable size, indicating that the algorithm might lack a good-quality lower bound in order to make further reductions.

4. Reduction Algorithm

The size of a QKP instance may be considerably reduced by using some reduction rules to fix variables at their optimal value. As mentioned in the introduction, we use aggressive reduction for this phase.

Assume that we have an incumbent solution x^* of value z. In its simplest form, the reduction algorithm branches on a given variable x_j and derives the upper bounds u_j^0 and u_j^1 corresponding to the two branches $x_j = 0$ and $x_j = 1$. If one of the branching nodes can be fathomed, e.g., since $u_j^0 \le z$, then we may fix the decision variable to the value that corresponds to the other branch, e.g., $x_j = 1$. We call this method *single-fix* since we fix variables by looking only one variable ahead.

In the generalized reduction, we look several variables ahead. So, in order to reduce variable x_{v_1}, we start a branching process with x_{v_1} at the root node, and branching up to a fixed depth d on variables $x_{v_2}, \ldots, x_{v_d}$. If all branches in one subtree of x_{v_1} can be fathomed, we may conclude that any improving solution must follow the opposite branch and hence can fix the variable at the corresponding value.

Figure 1 illustrates the principle: we assume that an incumbent solution of value $z = 10$ has been found at the root node, and that we try to reduce variable x_j. Upper bounds u are written inside each node. In the ordinary reduction we branch on $x_j = 0$ and $x_j = 1$, finding upper bounds 12 and 11 respectively, which means that we cannot fix x_j. In the generalized reduction, we start a branch and bound algorithm with x_j at the root node, which explores the search tree up to

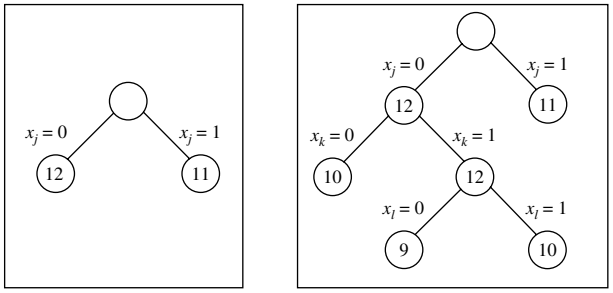

Figure 1 **Illustration of Ordinary Reduction (Left) and Generalized Reduction (Right)**

three levels of depth. Since all leaf nodes in the left search tree could be fathomed, we may conclude that x_j can be fixed at 1.

Our experimental results have shown that generalized reduction with a search-tree depth of two gives a good trade-off between reduction strength and computational time. Moreover, we restrict the reduction to consecutive pairs of variables x_j and x_{j+1}. For each pair of variables we have to evaluate four nodes, setting $(x_j, x_{j+1}) = (0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$, which can be done efficiently as follows. Assume that the current LP solution of the two variables is $x_j = \hat{x}_j$ and $x_{j+1} = \hat{x}_{j+1}$. We first evaluate the node $(x_j, x_{j+1}) = (1-\hat{x}_j, 1-\hat{x}_{j+1})$. If the node is not fathomed by the upper-bound test, we know due to monotonicity of the upper bound that neither of the three other nodes will be fathomed, and hence we continue with the next pair of variables. Otherwise we evaluate the node that sets $(x_j, x_{j+1}) = (1-\hat{x}_j, \hat{x}_{j+1})$ and fix x_j to $\hat{x}_j$ if the node is fathomed. In a similar way we evaluate the node setting $(x_j, x_{j+1}) = (\hat{x}_j, 1-\hat{x}_{j+1})$ and possibly fix x_{j+1} to $\hat{x}_{j+1}$. Note that the fourth node $(x_j, x_{j+1}) = (\hat{x}_j, \hat{x}_{j+1})$ does not need to be evaluated since it corresponds to the present solution.

We call the above reduction *double-fix*. It should be clear that double-fix dominates single-fix reduction. If variable x_j is fixed at any value we remove the corresponding row and column from the profit matrix. Moreover, if x_j is fixed at 1, we also increase diagonal entry p_{ii} by $p_{ij} + p_{ji}$, for $i \in N\backslash\{j\}$, and decrease c by w_j.

We use the two bounds U^2_{CPT} or U^2_{BFS} for the reduction. For each of the bounds we first compute the Lagrangian multipliers by solving the Lagrangian dual (6) and (10), and then for each variable try to fix it at its optimal value using the method described above.

When calculating the Lagrangian multipliers, we wish to compute them to such an accuracy that the resulting bound can be used to reduce variables. In order to determine if the multipliers have been calculated with sufficient accuracy, we reduce the instance using the suboptimal multipliers at different stages of the subgradient algorithm. If the instance cannot be reduced, the Lagrangian multipliers are improved further.

In the reduction algorithm we employ the strategy of trying to reduce the instance with the bounds that are the least computationally expensive first. By first reducing some variables using a fast bound, the instance is smaller when the computationally expensive bound is employed. This greatly diminishes the time used in the reduction phase.

We start by reducing the instance using the relatively fast bound U^2_{CPT}. If the reduction procedure fixes at least one variable, we apply subgradient optimization to the reduced problem, followed by a new reduction. If unfixed variables remain we reduce using U^2_{BFS}, increasing the group size n/m (and thereby the computation time required) after each pass. The group sizes used are 2, 4, and 8. Whenever reduction with the U^2_{BFS} bound succeeds in fixing some variables, the instance is again reduced using the U^2_{CPT} bound before proceeding. This is repeated until no variables can be fixed, and we proceed to branch and bound.

5. The Branch and Bound Algorithm

Our branch and bound algorithm is based on the upper-bounding procedure suggested by Caprara et al. (1999) and summarized in Section 2.1. At the root node of the branching tree, we apply subgradient optimization to find a good problem reformulation. The nodes of the branching tree other than the root are processed quite fast, 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), possibly updating the incumbent solution and applying branching.

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 for $\widehat{U}^2_{\text{CPT}}$.

As in the algorithm by Caprara et al. (1999) 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 that 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

$$\pi_i = p_{ii} + \max\Bigg\{ \sum_{j \in N\backslash\{i\}} (p_{ij} + p_{ji}) x_j : \sum_{j \in N\backslash\{i\}} w_j x_j \le c - w_i, \; 0 \le x_j \le 1, \; j \in N\backslash\{i\} \Bigg\}, \quad (11)$$

which represents an upper bound on the profit ob-

tained by setting variable x_i to 1. Notice that we make use of profits from rows as well as columns in (11) as opposed to Caprara et al. (1999), who used π_i analogous to (2) with i and j interchanged. The expression (11) is unaffected by the reformulation and hence gives a more correct estimate on the profit obtained by setting variable x_i to 1.

We reorder the variables according to nonincreasing values of π_i, and systematically branch on the variable with the smallest index among the unfixed ones. The branching scheme follows the framework of the algorithm by Caprara et al. (1999), to which the reader is referred for more details.

The upper-bound computation is the most time-consuming operation at any node of the branching tree. It is therefore extremely important to have a very efficient implementation of this part in order to limit the overall computing time. Using appropriate data structures Caprara et al. (1999) showed that if the upper bound is calculated by the LP relaxation of (3) then the following holds:

Proposition 1. *Every node of the branching tree is processed in linear* (*in n*) *expected time.*

As we derive the upper bound in each node by solving (3) as an integer knapsack problem, the above proposition does not hold for our implementation. The n continuous knapsack problems corresponding to the LP relaxation of (2) can, however, be solved in linear expected time.

6. Computational Experiments

All tests where conducted using gcc 3.2.2 on an Intel Pentium4 processor at 2.4 GHz. A time limit of 12 hours was imposed for each instance. In the following sections we consider both standard QKP instances from the literature as well as a number of new instances adapted from various graph-theoretical problems.

6.1. Standard Instances

Gallo et al. (1980) presented a number of randomly generated instances for the QKP. These instances have been used as standard benchmark tests by most subsequent authors, including Billionnet and Calmels (1996), Michelon and Veilleux (1996), and Caprara et al. (1999).

Let Δ be the *density* of the instance, i.e., the percentage of nonzero 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 et al. (1980) actually chose the capacities in $[1, \sum_{j=1}^{n} w_j]$ but later papers have increased the lower limit.)

In the following tables we consider instances with densities $\Delta = 25\%$, 50%, 75%, and 100% for size n up to 1,500 in multiples of 100. For each size and density we have constructed and solved ten instances. The entries in the table are average values over the instances that could be solved within the time limit.

Table 1 Number of Instances Solved Out of Ten, Standard Instances

$n\backslash\delta$	25	50	75	100
100	10	10	10	10
200	9	10	10	10
300	10	10	10	10
400	9	10	10	10
500	9	10	10	10
600	9	10	10	10
700	6	10	10	10
800	6	10	10	10
900	6	10	10	10
1,000	6	10	10	10
1,100	2	9	10	10
1,200	5	9	10	9
1,300	1	9	10	8
1,400	4	10	9	9
1,500	3	6	10	10
Avg.	6.3	9.5	9.9	9.7

Table 1 reports the number of instances solved. For density 50% and above, most instances were solved. As previously noticed by Caprara et al. (1999), it seems that the low-density problems are the most difficult to solve. Although the upper bound by Billionnet et al. (1999) is quite tight for low-density problems, it is difficult to find lower bounds of high quality for these problems. Moreover the reduction scheme does not work so well for low-density problems, as the upper bound is not changed much by fixing one or two variables.

In Table 2, the average computation times are reported. Most instances are solved within a few hours of CPU time on average, which is reasonable for these

Table 2 Total Computation Time in Seconds (Average of Solved Instances), Standard Instances

$n\backslash\delta$	25	50	75	100
100	210.7	54.2	6.7	2.7
200	860.0	168.9	23.0	76.5
300	4,031.9	556.8	94.7	90.3
400	1,190.1	978.3	295.0	173.2
500	3,865.7	1,982.8	266.6	392.2
600	5,565.4	3,711.1	744.5	794.7
700	10,422.0	3,158.7	1,471.6	575.5
800	2,916.5	2,939.4	1,768.4	1,375.2
900	20,440.0	2,461.9	2,585.2	971.7
1,000	13,281.9	8,336.6	2,419.7	1,809.9
1,100	8,090.1	5,692.1	3,929.6	2,263.6
1,200	446.1	11,544.4	4,093.2	5,002.3
1,300	3,393.8	15,948.2	6,644.5	3,386.3
1,400	6,717.0	22,606.5	10,947.7	5,416.3
1,500	5,077.9	19,135.8	12,770.7	12,771.1
Avg.	5,767.3	6,618.4	3,204.1	2,340.1

Table 3 Time Used by Reduction Phase (Average of Solved Instances), Standard Instances

$n\backslash\delta$	25	50	75	100
100	189.2	54.2	6.7	2.7
200	584.0	168.7	23.0	76.5
300	3,207.8	556.7	94.7	90.1
400	1,189.9	975.6	294.8	173.0
500	3,864.6	1,982.7	266.6	391.9
600	5,564.8	3,611.3	743.5	794.5
700	10,421.7	3,141.9	1,470.0	575.2
800	2,916.3	2,938.6	1,764.5	1,374.6
900	20,439.8	2,461.6	2,579.2	971.3
1,000	13,281.1	8,298.9	2,419.6	1,740.8
1,100	8,089.9	5,690.1	3,924.0	2,246.9
1,200	446.0	11,538.6	4,076.0	4,052.8
1,300	3,393.6	15,939.7	6,642.0	3,385.4
1,400	6,714.8	22,512.9	8,305.5	5,412.7
1,500	4,919.9	19,115.6	12,755.1	12,414.4
Avg.	5,681.6	6,599.1	3,024.3	2,246.9

Table 5 Objective Value (Average of Solved Instances), Standard Instances

$n\backslash\delta$	25	50	75	100
100	43,631.2	91,887.9	192,668.3	260,186.2
200	193,798.8	561,313.4	448,942.6	887,270.1
300	599,935.6	1,093,950.3	1,968,163.6	2,546,920.8
400	1,408,773.4	2,140,089.4	3,298,495.0	4,083,908.0
500	1,684,972.3	2,987,465.9	4,402,497.5	7,078,125.3
600	3,488,900.1	4,084,686.4	8,384,937.0	8,475,661.0
700	2,661,969.2	7,588,377.9	8,654,936.4	11,320,611.2
800	3,866,772.2	6,219,035.0	11,137,681.6	19,685,467.6
900	5,155,890.5	9,998,054.0	9,526,540.9	16,137,230.3
1,000	8,886,079.3	14,464,274.4	17,338,565.3	25,372,726.9
1,100	3,644,757.5	15,561,306.1	22,334,441.4	43,524,304.8
1,200	15,216,608.0	19,180,365.4	32,351,394.2	40,863,251.4
1,300	1,389,218.0	18,301,520.0	34,343,179.1	27,058,156.2
1,400	21,651,174.5	26,857,966.2	34,111,466.4	58,407,674.9
1,500	24,252,955.0	23,208,258.3	42,625,118.1	37,499,235.4
Avg.	6,276,362.4	10,155,903.4	15,407,935.2	20,213,382.0

large instances. Table 3 shows the corresponding time used in the reduction phase. By comparing Tables 2 and 3 we notice that only a minor amount of time is spent after the reduction phase, i.e., inside the branch and bound algorithm. The reason for this behavior is that the aggressive reduction phase generally is able to reduce the instance to a very small size, leaving little work to do for the branch and bound algorithms. This can also be seen from Table 4, which shows the average number of items remaining after the reduction phase. On average, the reduced instances contain well below 100 items, which is much more manageable for the branch and bound algorithm. Table 5 reports the found solution values for future comparison.

Finally, we compare the present algorithm with that of Caprara et al. (1999) in Table 6. The two algorithms basically contain the same branch and bound algorithm, but differ in the reduction part (upper and lower bounds, iterations of Lagrangian multipliers, reduction). The computation times of this algorithm are shown only when all ten instances could be solved within the time limit of 12 hours, so a missing entry does necessarily mean that that no instances were solved. The test setup is the same as previously described. The Caprara-Pisinger-Toth algorithm solves small instances faster, since it makes use of a much simpler reduction algorithm. However, it is not able to solve large instances, and in particular low-density problems can be solved only for slightly more than 100 items. These experiments illustrate that reduction is a key ingredient in solving difficult $\mathcal{NP}$-hard optimization problems.

Table 4 Size of Instance After Reduction Phase (Average of Solved Instances), Standard Instances

$n\backslash\delta$	25	50	75	100
100	30.6	5.5	2.8	3.1
200	28.2	8.2	2.4	8.4
300	56.5	14.2	3.4	20.1
400	24.6	40.4	17.4	10.0
500	46.9	11.2	6.9	11.8
600	34.0	19.7	28.3	16.3
700	31.2	63.2	33.1	16.5
800	26.5	49.5	73.5	32.5
900	23.7	11.0	22.3	18.5
1,000	37.3	46.2	17.3	28.1
1,100	39.0	24.7	43.1	28.0
1,200	21.2	37.1	47.6	77.6
1,300	45.0	57.6	58.2	40.5
1,400	58.0	125.7	58.8	28.2
1,500	120.7	39.2	44.6	56.8
Avg.	41.6	36.9	30.6	26.4

Table 6 Total Computation Time in Seconds, Caprara, Pisinger, Toth (Average of Ten Instances)

$n\backslash\delta$	25	50	75	100
100	18.7	2.1	0.4	0.4
200	—	11.6	2.8	20.5
300	—	—	20.4	22.1
400	—	—	—	587.2

6.2. New Instances

To get a broader profile of the presented algorithm, we also consider some instances adapted from various related problem, including dispersion problems, knapsack dispersion problems, and dense subgraph problems. All these problems are easily formulated as QKP problems, as described below.

6.2.1. Dispersion Problems. The dispersion problem is that of locating q facilities at n possible locations to maximize the total distance sum. If we let d_{ij} denote the distance between two locations, and x_j be

a binary variable that indicates whether facility j is chosen, we may formulate the problem as

$$\begin{aligned} \text{maximize} \quad & \sum_{i\in N}\sum_{j\in N} d_{ij}x_i x_j \\ \text{subject to} \quad & \sum_{j\in N} x_j = q \\ & x_j \in \{0,1\}, \quad j \in N. \end{aligned} \tag{12}$$

The dispersion problem can be trivially transformed to a QKP by setting all weights in the QKP to $w_j := 1$, profits are set to $p_{ij} := d_{ij}$, while the capacity is set to $c := q$. In the following we will consider a number of instances presented in Pisinger (2006):

- GEO The *geometrical problems* reflects a typical location problem in the Euclidean space, as presented in Erkut (1990). The n facilities are randomly located in a 100×100 rectangle, and d_{ij} is the Euclidean distance between facilities i and j.
- WGEO The *weighted geometrical problems* should reflect some kind of weighting of the distances (Pisinger 2006). As previously, the n facilities are randomly located in a 100×100 rectangle, and each facility j is assigned a weight α_j in the interval $[5, \ldots, 10]$. The distance d_{ij} is then found as $\alpha_i\alpha_j$ times the Euclidean distance between facility i and j.
- EXP Instances with *exponential distribution* of the distances were presented in Kincaid (1992). Each d_{ij} with $i < j$ is randomly drawn from an exponential distribution with mean value 50. We set $d_{ij} = d_{ji}$ to ensure symmetry.
- RAN Instances with *random distances* are generated with d_{ij} randomly distributed in $[1, \ldots, 100]$.

In all the instances, we set $d_{ii} = 0$ for $i = 1, \ldots, n$. The number of facilities q is randomly chosen in the interval $[2, \ldots, n-2]$.

6.2.2. Knapsack Dispersion Problems. The dispersion problems presented in the previous section may be generalized to knapsack-like problems by choosing w_j randomly in $[1, \ldots, 100]$. This leads to the following instances:

- KP-GEO *Knapsack geometrical problems*: profits p_{ij} are as in GEO.
- KP-WGEO *Knapsack weighted geometrical problems*: profits p_{ij} are as in WGEO.
- KP-EXP *Knapsack exponential problems*: profits p_{ij} are as in EXP.
- KP-RAN *Knapsack random problems*: profits p_{ij} are as in RAN.

The capacity is in each instance chosen as $\frac{1}{2}\sum_{j=1}^{n} w_j$.

6.2.3. Dense Subgraph Problems. Given a graph $G = (V, E)$, the dense subgraph problem chooses a subset of nodes $U \subseteq V$ of cardinality q for which the imposed graph contains the most possible edges. This can be formulated as

$$\begin{aligned} \text{maximize} \quad & \sum_{i\in N}\sum_{j\in N} d_{ij}x_i x_j \\ \text{subject to} \quad & \sum_{j\in N} x_j = q \\ & x_j \in \{0,1\}, \quad j \in N, \end{aligned} \tag{13}$$

where $d_{ij} = 1$ if edge $(i, j) \in E$, and $d_{ij} = 0$ otherwise. We consider the following four instance types:

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

In all the instances, we set $d_{ii} = 0$ for $i = 1, \ldots, n$. The number of nodes q is randomly chosen in the interval $[2, \ldots, n-2]$. For technical reasons all values of d_{ij} have been scaled by a factor of 100 to obtain solution values of a magnitude comparable to the other instances.

6.2.4. Results. The new test instances have been run for $n = 25$, 50, 100, 200, 400, 800, and 1,600. For each size n and instance type, we generated and solved ten instances. It was not possible to generate WGEO and KP-WGEO instances of size $n = 1{,}600$ as the distances are 1–2 orders of magnitude larger than for the other problems, and hence caused overflow for the largest instances.

As for the standard QKP instances, we report the number of solved instances out of ten in Table 7, the total solution time in Table 8, the time used by the reduction phase in Table 9, and finally the size of the instance after reduction in Table 10.

For the dispersion problems GEO, WGEO, EXP, and RAN, we are able to solve a few instances of size up to $n = 1{,}600$ but generally we cannot expect to solve all ten instances for large values of n. The GEO and WGEO instances seem to be the easiest to solve, while the EXP instances tend to be more difficult.

The knapsack dispersion problems are easier to solve. This may be due to the fact that larger variation in the weights make it easier to reduce the instances, as seen from Table 10. We are able to solve nearly all the considered instances to optimality in reasonable time.

Finally, the dense subgraph problems are the most difficult to solve as there is very little variation in the

Table 7 Number of Instances Solved Out of Ten, New Instances

n	GEO	WGEO	EXP	RAN	KP-GEO	KP-WGEO	KP-EXP	KP-RAN	DSUB25	DSUB50	DSUB75	DSUB90
25	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0
50	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0
100	6.0	10.0	4.0	3.0	10.0	10.0	10.0	10.0	5.0	5.0	5.0	5.0
200	5.0	10.0	0.0	2.0	10.0	10.0	10.0	10.0	2.0	2.0	2.0	4.0
400	4.0	2.0	0.0	1.0	10.0	10.0	10.0	10.0	1.0	1.0	0.0	0.0
800	3.0	0.0	0.0	0.0	10.0	10.0	10.0	10.0	2.0	0.0	0.0	1.0
1,600	1.0	—	0.0	0.0	10.0	—	10.0	8.0	0.0	0.0	0.0	0.0

Table 8 Total Computation Time in Seconds (Average of Solved Instances), New Instances

n	GEO	WGEO	EXP	RAN	KP-GEO	KP-WGEO	KP-EXP	KP-RAN	DSUB25	DSUB50	DSUB75	DSUB90
25	0.4	1.1	2.4	1.9	0.1	0.1	0.1	0.1	1.4	2.1	1.2	0.5
50	7.8	9.4	10.9	23.3	0.4	0.4	0.4	0.7	7.1	50.7	199.3	9.3
100	7,436.6	296.0	1,000.3	788.3	1.8	1.6	5.9	1.7	928.4	3,624.4	107.4	14.6
200	2,875.4	1,187.5	—	3,161.4	9.8	9.8	69.3	20.4	664.7	133.4	104.9	193.3
400	5,455.4	3,008.7	—	546.2	57.9	49.2	204.5	198.2	231.0	547.1	—	—
800	1,312.8	—	—	—	350.6	326.1	1,810.7	1,404.0	3,107.5	—	—	19,387.1
1,600	13,208.1	—	—	—	3,555.8	—	10,340.3	10,788.0	—	—	—	—

Table 9 Time Used by Reduction Phase (Average of Solved Instances), New Instances

n	GEO	WGEO	EXP	RAN	KP-GEO	KP-WGEO	KP-EXP	KP-RAN	DSUB25	DSUB50	DSUB75	DSUB90
25	0.3	1.1	2.4	1.8	0.1	0.1	0.1	0.1	1.3	2.0	1.1	0.5
50	6.4	9.2	6.0	4.4	0.4	0.4	0.4	0.7	3.8	3.8	3.8	6.5
100	67.4	294.7	67.9	121.1	1.8	1.6	5.9	1.7	112.6	131.7	48.3	14.2
200	290.4	911.1	—	306.3	9.8	9.8	69.1	20.4	318.9	132.2	104.8	192.3
400	433.9	2,424.3	—	545.8	57.9	49.2	204.3	198.1	231.0	538.5	—	—
800	1,312.7	—	—	—	350.6	326.1	1,809.1	1,403.4	3,104.6	—	—	19,286.0
1,600	13,207.5	—	—	—	3,555.8	—	10,339.8	10,785.2	—	—	—	—

profits and weights. It seems that denser problems are easier to solve, but this needs to be confirmed by more tests.

The tests indicate that the presented reduction algorithms perform best if there is some variation in the weights w_i.

7. Conclusions

The QKP is an important model with numerous applications. Due to its intrinsic difficulty, it has attracted much interest in the last decade. Much of the research develops methods where the goal is general techniques that can be applied to a large variety of related problems.

The QKP is a difficult $\mathcal{NP}$-hard optimization problem, in the sense that existing branch and bound methods are not able to solve large instances. For such problems, reduction is a key ingredient in the solution process. We have presented a framework called *aggressive reduction* in which preprocessing techniques become the main solution approach, while branch and bound is a subordinate method. Computational experiments for the QKP, where aggressive reduction is compared to "normal" reduction and branch and bound, shows that the framework has great potential.

Table 10 Size of Instance After Reduction Phase (Average of Solved Instances), New Instances

n	GEO	WGEO	EXP	RAN	KP-GEO	KP-WGEO	KP-EXP	KP-RAN	DSUB25	DSUB50	DSUB75	DSUB90
25	13.4	6.4	6.3	16.6	2.0	1.7	2.4	1.7	13.4	13.6	13.0	9.4
50	39.9	28.7	40.1	43.7	1.9	1.3	2.9	2.8	37.5	35.7	34.2	9.1
100	55.8	92.8	50.5	95.0	0.7	2.6	3.0	1.5	60.0	49.8	28.8	13.0
200	50.2	140.2	—	121.5	3.4	5.0	13.7	4.8	100.0	46.0	13.5	18.3
400	51.0	400.0	—	47.0	2.7	2.1	19.1	13.5	2.0	88.0	—	—
800	18.0	—	—	—	2.2	2.9	46.5	26.7	103.5	—	—	800.0
1,600	62.0	—	—	—	1.5	—	23.4	23.4	—	—	—	—

In many instances, aggressive reduction reduces the instance to a trivial or at least manageable size.

Regarding concrete contributions to the QKP, we should mention the solution of a core problem for finding improved lower bounds, tighter variants of the U^2_{CPT}, improved methods for finding Lagrangian multipliers associated with the bounds U^2_{BFS} and U^2_{CPT}, and finally better branching rules.

From a practical point of view, the developed algorithm is able to solve standard instances of considerably larger size than previously reported in the literature. A majority of the standard instances with up to 1,500 variables has been solved, where previous approaches (Caprara et al. 1999) solved instances with up to 400 variables. An even more important achievement is that the new techniques also work well for medium- and low-density instances, where previous approaches worked mainly for high-density instances.

The presented algorithm has also been successfully applied to a number of dispersion problems, where in particular the results on knapsack dispersion problems are very promising.

Acknowledgments

The authors thank the August Krogh Institute and Zoological Museum, University of Copenhagen for having made the *BioCluster* supercomputer available for these experiments. They also thank two anonymous referees for their valuable comments.

References

Balas, E., E. Zemel. 1980. An algorithm for large zero-one knapsack problems. *Oper. Res.* **28** 1130–1154.

Billionnet, A., F. Calmels. 1996. Linear programming for the 0-1 quadratic knapsack problem. *Eur. J. Oper. Res.* **92** 310–325.

Billionnet, A., A. Faye, E. Soutif. 1999. A new upper-bound and an exact algorithm for the 0-1 quadratic knapsack problem. *Eur. J. Oper. Res.* **112** 664–672.

Caprara, A., D. Pisinger, P. Toth. 1999. Exact solution of the quadratic knapsack problem. *INFORMS J. Comput.* **11** 125–137.

Chaillou, P. P. Hansen, Y. Mahieu. 1989. Best network flow bound for the quadratic knapsack problem. B. Simeone, ed. *Combinatorial Optimization, Lecture Notes in Mathematics*, Vol. 1403. Springer Verlag, Heidelberg, Germany, 225–235.

Cook, W., P. Seymour. 2003. Tour mergining via branch-decomposition. *INFORMS J. Comput.* **15** 233–248.

Erkut, E. 1990. The discrete *p*-dispersion problem. *Eur. J. Oper. Res.* **46** 48–60.

Ferreira, C. E., A. Martin, C. C. de Souza, R. Weismantel, L. A. Wolsey. 1996. Formulations and valid inequalities for node capacitated graph partitioning. *Math. Programming* **74** 247–266.

Gallo, G., P. L. Hammer, B. Simeone. 1980. Quadratic knapsack problems. *Math. Programming Stud.* **12** 132–149.

Hammer, P. L., D. J. Rader. 1997. Efficient methods for solving quadratic 0-1 knapsack problems. *INFOR* **35** 170–182.

Helmberg, C., F. Rendl, R. Weismantel. 1996. Quadratic knapsack relaxations using cutting planes and semidefinite programming. W. H. Cunningham, S. T. McCormick, M. Queyranne, eds. *Proc. Fifth IPCO Conf.*, Vol. 1084. Springer Verlag, Heidelberg, Germany, 175–189.

Johnson, E. L., A. Mehrotra, G. L. Nemhauser. 1993. Min-cut clustering. *Math. Programming* **62** 133–152.

Kellerer, H., U. Pferschy, D. Pisinger. 2004. *Knapsack Problems*. Springer Verlag, Heidelberg, Germany.

Kincaid, R. K. 1992. Good solutions to discrete noxious location problems via metaheuristics. *Ann. Oper. Res.* **40** 265–281.

Michelon, P., L. Veilleux. 1996. Lagrangean methods for the 0-1 quadratic knapsack problem. *Eur. J. Oper. Res.* **92** 326–341.

Park, K., K. Lee, S. Park. 1996. An extended formulation approach to the edge-weighted maximal clique problem. *Eur. J. Oper. Res.* **95** 671–682.

Pisinger, D. 2006. Upper bounds and exact algorithms for *p*-dispersion problems. *Comput. Oper. Res.* **33** 1380–1398.

Polzin, T., S. Vahdati. 2002. Extending reduction techniques for the Steiner tree problem: A combination of alternative-and bound-based approaches. R. H. Möhring, R. Raman, eds. *Algorithms—ESA 2002, Lecture Notes in Computer Science*, Vol. 2461. Springer Verlag, Heidelberg, Germany, 795–807.

Rasmussen, A. B., R. Sandvik. 2002. Kvaliteten af grænsev ærdier for det kvadratiske knapsack problem. Working Paper Project 02-09-7, (D. Pisinger, supervisor), DIKU, University of Copenhagen, Copenhagen, Denmark.

Rhys, J. 1970. A selection problem of shared fixed costs and network flows. *Management Sci.* **17** 200–207.

Thomadsen, T., J. Larsen. 2007. A hub location problem with fully interconnected backbone and access networks. *Comput. Oper. Res.* **34** 2520–2531.

Witzgall, C. 1975. Mathematical methods of site selection for electronic message systems (EMS). Technical Report, National Bureau of Standards, Gaithersburg, MD.