# Solution of Large-sized Quadratic Knapsack Problems Through Aggressive Reduction

David Pisinger, Anders Bo Rasmussen and Rune Sandvik

Technical Report no. 2004/11
ISSN: 0107-8283
CR Subject Classification:

# Solution of Large-sized Quadratic Knapsack Problems Through Aggressive Reduction[*]

David Pisinger, Anders Bo Rasmussen, Rune Sandvik
DIKU, Univ. of Copenhagen, Univ.parken 1, Copenhagen, Denmark
e-mail: pisinger@diku.dk, fuzz@fuzz.dk, mail@runesandvik.dk

October 2003

**Abstract**

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 1500 binary variables can be reduced to less 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.

**Keywords**: 0-1 Quadratic Programming, Knapsack Problem, Aggressive Reduction, Lagrangian Relaxation, Branch-and-Bound.

[*]Tech. Report 2004, DIKU, University of Copenhagen, Denmark

## 1 Introduction

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

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

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 [18], location problems [17], compiler design [11, 10], VLSI design [7]. Moreover numerous graph problems can be formulated as QKP, e.g. the dense subgraph problem, the weighted maximum $b$-clique problem [14], and QKP appears when solving the graph partitioning problem [11].

Numerous upper bounds have been presented for the QKP. A family of upper bounds based on upper planes were presented by Gallo, Hammer Simeone [8], while a bound based on Lagrangian relaxation of the capacity constraint was presented by Chaillou, Hansen, Mahieu [5]. Caprara, Pisinger, Toth [4] presented upper bonds based on Lagrangian relaxation and reformulation. Billionnet, Calmels [2] presented a bound based on Linearisation. Bounds based on Lagrangian decomposition have been presented by Michelon Veuilleux [13] and Billionnet, Faye, Soutif [3]. For a recent survey on knapsack problems and also the corresponding upper bounds see Kellerer, Pferschy, Pisinger [12]. A comprehensive experimental comparison of the above bounds is presented in Rasmussen and Sandvik [16].

Several reduction methods, used for fixing some of the decision variables at their optimal value, have been presented for the QKP, see e.g. [9, 4]. Recent advances in the solution of difficult $\mathcal{NP}$-hard optimization problems have shown the importance of good reduction techniques, see e.g. the seminar work of Polzin and Vahdati [15] 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 are 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 of tight upper and lower bounds in a reduction algorithm. In aggressive reduction, numerous bounds are used. If one bound fails in reducing a variables, the algorithm should not give up, but instead try to use other bounds for the reduction. The last 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 which is repeated as long as at least one variable was reduced. Two different upper bounds are used for the reduction: The upper bound $U^2_{\text{CPT}}$ by Caprara, Pisinger and Toth [4] based on upper planes and $U^2_{\text{BFS}}$ by Billionnet, Faye and Soutif [3] based on Lagrangian decomposition.

As observed in the computational study by Rasmussen and Sandvik [16], the upper bound $U^2_{\text{CPT}}$ has low computation time and good quality for randomly generated instances with high density of the profit matrix. The slower upper bound $U^2_{\text{BFS}}$ returns on average tighter upper bounds than $U^2_{\text{CPT}}$, especially on low density instances. We alternate between using $U^2_{\text{CPT}}$ and $U^2_{\text{BFS}}$, using the fast bound $U^2_{\text{CPT}}$ to reduce the instance to a manageable size, before employing the slower bound $U^2_{\text{BFS}}$. An additional motivation for alternating between the bounds is that $U^2_{\text{BFS}}$ does not dominate $U^2_{\text{CPT}}$ as shown in [16]. To further strengthen the $U^2_{\text{CPT}}$ bound, 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 technique, and the solution of a core problem based on a merging of sub-optimal 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, Pisinger, Toth [4]. 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 gives 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 used 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 the aggressive reduction algorithm which is used to fix most possible variables at their optimal values, before proceeding to the branch-and-bound part, described in Section 5. Finally, Section 6 presents a number of computational experiments with the developed algorithm solving instances with up to 1500 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 of the bound suggested by Caprara, Pisinger, Toth [4] 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, Faye and Soutif [3] in the reduction phase.

### 2.1 Caprara, Pisinger, Toth

The bound by Caprara, Pisinger, Toth [4] 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

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

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

$$\begin{aligned} \text{maximize} \quad & \sum_{j\in N} \pi_j x_j \\ \text{subject to} \quad & \sum_{j\in N} w_j x_j \le c \\ & x_j \in \{0,1\}, \quad j \in N. \end{aligned} \tag{3}$$

If we relax the integrality constraints in the above subproblems we obtain the bound $U^{1*}_{\text{CPT}}$ 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, Pisinger, Toth [4] further strengthened the bound by noting that the objective function can be reformulated as

$$\sum_{i\in N}\sum_{j\in N} p_{ij}x_i x_j = \sum_{i\in N}\sum_{j\in N}(p_{ij}+\lambda_{ij})x_i x_j \tag{4}$$

for any skew-symmetric matrix $\Lambda$ (i.e. $\lambda_{ij} = -\lambda_{ji}$). Let QKP($\Lambda$) denote the problem

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

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

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

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

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

## 2.2 Billionnet, Faye, Soutif

The bound by Billionnet, Faye and Soutif [3] is based on a partitioning of $N$ into $m$ disjoint classes $\{I_1,\ldots,I_m\}$ satisfying $\cup_{k=1}^m I_k = N$. The main idea in the bound by Billionnet, Faye, Soutif 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 which may be solved in $O(n)$ time.

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

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

Using Lagrangian decomposition we introduce binary copy variables $y_j^k = x_j$ for $j \in N\setminus I_k$ and add $k$ redundant capacity constraints. Moreover we add copy constraints concerning the quadratic terms $x_ix_j = x_iy_j^k$, getting the following formulation. The function $\mathrm{set}(i)$ 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 $\mathrm{set}(i)$ is well defined. Hence, we get the formulation

$$\begin{array}{lll} \text{maximize} & \sum_{k=1}^{m}\left(\sum_{i\in I_k}\sum_{j\in I_k} p_{ij}x_ix_j + \sum_{i\in I_k}\sum_{j\in N\setminus I_k} p_{ij}x_iy_j^k\right) & \\ \text{subject to} & x_j = y_j^k & k=1,\dots,m,\ j\in N\setminus I_k \\ & x_iy_j^{\mathrm{set}(i)} = x_jy_i^{\mathrm{set}(j)} & i\in N,\ j\in N, \mathrm{set}(i)\neq\mathrm{set}(j) \\ & \sum_{i\in I_k} w_ix_i + \sum_{j\in N\setminus I_k} w_jy_j^k \le c & k=1,\dots,m \\ & x_i\in\{0,1\} & i\in I_k, k=1,\dots,m \\ & y_j^k\in\{0,1\} & k=1,\dots,m,\ j\in N\setminus I_k \end{array} \qquad (7)$$

Lagrangian relaxing the two first constraints using multipliers $\Lambda = \{\lambda_j^k\}$, $k=1,\dots,m, j\in N\setminus I_k$ respectively $M = \{\mu_{ij}\}$, $i\in N, j\in N, \mathrm{set}(i)\neq\mathrm{set}(j)$ we can decompose (7) into $m$ independent problems (see [3] or [12] for details). The $m$ problems $L_k(I_k,\Lambda,M), k=1,\dots,m$ are on the form

$$\begin{array}{lll} \text{maximize} & \sum_{i\in I_k}\sum_{j\in I_k} p_{ij}x_ix_j + \sum_{i\in I_k}\left(\sum_{h\neq k}\lambda_i^h\right)x_i + & \\ & \sum_{j\in N\setminus I_k}\left(\sum_{i\in I_k} p_{ij}x_i\right)y_j^k - \sum_{j\in N\setminus I_k}\lambda_j^ky_j^k + \sum_{j\in N\setminus I_k}\left(\sum_{i\in I_k}(\mu_{ij}-\mu_{ji})x_i\right)y_j^k & \\ \text{subject to} & \sum_{i\in I_k} w_ix_i + \sum_{j\in N\setminus I_k} w_jy_j^k \le c & \\ & x_i\in\{0,1\} & i\in I_k, k=1,\dots,m \\ & y_j^k\in\{0,1\} & j\in N\setminus I_k \end{array} \qquad (8)$$

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 $\tilde{p}_j = \sum_{i\in I_k} p_{ij}x_i - \lambda_j^k + \sum_{i\in I_k}(\mu_{ij}-\mu_{ji})x_i$ hence getting $L_k(I_k,\Lambda,M)$

in the form

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

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

$$U^2_{\text{BFS}} = \min_{\Lambda, M} U^1_{\text{BFS}}(\Lambda, M) \tag{10}$$

A sub-optimal choice of Lagrangian multipliers $\Lambda$ and $M$ can be found through subgradient optimization, leading to the bound $\hat{U}^2_{\text{BFS}}$ with $U^2_{\text{BFS}} \leq \hat{U}^2_{\text{BFS}} \leq U^1_{\text{BFS}}(0,0)$.

In order to make the computation of $U^1_{\text{BFS}}(\Lambda, M)$ 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 $\Lambda, M$ 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 $\lambda, \mu$ 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 used for iterating the Lagrangian multipliers in the bound $\hat{U}^2_{\text{BFS}}$. In their computational experiments, Billionnet, Faye, Soutif 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 depend 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 that result 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 $j = N \setminus \{I_k\}$. 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 [2]. 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 $\delta_j$ be the objective function increase when $x_j$ is set to 1. Otherwise, let $\delta_j$ 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. In every second step in the subgradient algorithm in the bound suggested by Caprara, Pisinger, Toth 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 only legal if $w_j - w_i + \sum_{\ell \in S} w_\ell \leq c$. Define the change in the objective function by applying the move $(i, j)$ by $\delta_{ij}$. 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

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

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 having completed 5000 iterations.

Both the local exchange heuristic and tabu search based heuristic only search solutions 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 only runs 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 which 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 [6], and somehow also the the ordinary core problem for a Knapsack Problem as presented by Balas and Zemel [1].

To be more specific, the core of the QKP is defined as follows. During the calculations 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 $f_j^0$ 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. Among the variables with $f_j^1 \approx f_j^0$

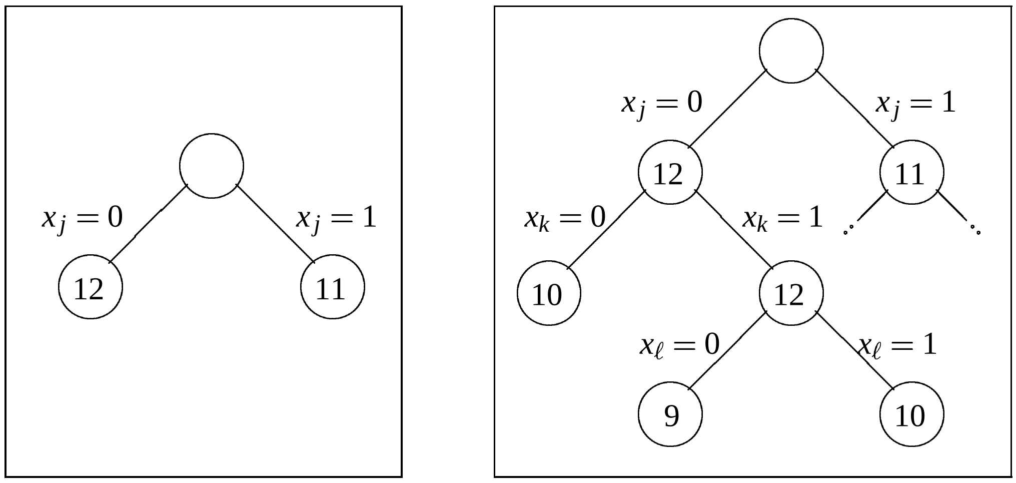

Figure 1: Illustration of ordinary (left) and generalized (right) reduction. We assume that an incumbent solution of value $z = 10$ has been found at the root node, and that we try to reduce variable $x_j$. Upper bounds $u$ are written inside each node. In the ordinary reduction we branch on $x_j = 0$ and $x_j = 1$ finding upper bounds 12 and 11 respectively, which means that we cannot fix $x_j$ to any of the values. 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 three levels of depth. Since all nodes in the left search tree could be fathomed we may conclude that $x_j$ can be fixed at 1.

we choose $n/4$ variables with $f_j^1$ closest to $f_j^0$ which define the core $C$ of the problem. 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 fairly computationally expensive, and is therefore only used 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 reduce further.

## 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 $u_j^0 \leq z$, then we may fix the decision variable to the value which corresponds to the other branch, e.g. $x_j = 1$. We call this method *single fix* since we fix variables by only looking one variable ahead.

In the generalized reduction, we may look more nodes ahead in the reduction. 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 $x_{v_2}, \dots, x_{v_d}$ (see Figure 1 for an example). If all branches in one subtree 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.

Our experimental results have shown that generalized reduction with a depth of the search tree of two, gives a good trade-off between reduction strength and computational time. We try to reduce the variable $x_j$ by first branching on $x_j$ and then on $x_{j+1}$. Variable $x_{j+1}$ is then reduced by branching on $x_{j+1}$ and then $x_j$. A few modifications to the implementation of enumerative reduction have been made. Firstly, we do not calculate bounds at level one in the branching tree. That is, we only consider the four subproblems corresponding to the leaves in a level two branching tree. Secondly, we take advantage of the fact that the four subproblems for reducing variable $x_j$ overlap with the four subproblems for reducing variable $x_{j+1}$. e.g when reducing variable $x_j$ we calculate the bound with $x_j = 0$ and $x_{j+1} = 1$ and when reducing $x_{j+1}$ the bound is also calculated with $x_{j+1} = 1$ and $x_j = 0$. This effectively halves the number of bound calculations required, to reduce variables $x_j$ and $x_{j+1}$. We call this reduction *double fix*. It should be clear that double fix dominates single fix reduction.

In the general case with a depth of the search tree of $m$ one may consider clusters of items $x_j, x_{j+1}, \dots, x_{j+m-1}$ to branch on. Due to the overlap of subproblems when reducing variables $x_j, x_{j+1}, \dots, x_{j+m-1}$, we only need to derive the $2^m$ upper bounds at the leave nodes once. On average $2^m/m$ bounds are derived for each item.

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 $p_{ii}$ by $p_{ij} + p_{ji}$, for $i \in N \setminus \{j\}$, and decrease $c$ by $w_j$.

We use the two bounds $U^2_{\text{CPT}}$ or $U^2_{\text{BFS}}$ 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 can not 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 $U^2_{\text{CPT}}$. 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 $U^2_{\text{BFS}}$, 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 $U^2_{\text{BFS}}$ bound succeeds in fixing some variables, the instance is again reduced using the $U^2_{\text{CPT}}$ 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, Pisinger, Toth 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.

We next describe in detail each part of the branch-and-bound algorithm outlined above. Let $N$ denote the set of variables that were not fixed by the reduction procedure. Moreover, let $\hat{P} = (\hat{p}_{ij})$ be the Lagrangian profit matrix associated with the best upper bound found by the subgradient procedure for $\hat{U}^2_{\text{CPT}}$.

Like in the algorithm by Caprara, Pisinger, Toth [4] 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

$$\pi_i = p_{ii} + \max\left\{ \sum_{j\in N\setminus\{i\}} (p_{ij} + p_{ji})x_j : \sum_{j\in N\setminus\{i\}} w_j x_j \le c - w_i,\ 0 \le x_j \le 1, j \in N \setminus \{i\} \right\} \tag{11}$$

which represents an upper bound on the profit obtained 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 [4] which used $\pi_i$ analogous to (2) just with $i$ and $j$ interchanged. The expression (11) is unaffected by the reformulation and hence give a more correct estimate on the profit obtained by setting variable $x_i$ to 1.

We reorder the variables according to non-increasing values of $\pi_i$, 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, Pisinger, Toth [4], hence we refer to this paper for a more detailed description.

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, Pisinger, Toth [4] showed that if the upper bound is calculated by the LP-relaxation of (3) then the following hold:

**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

We considered the classical classes of QKP instances presented in the literature. Gallo, Hammer and Simeone [8] solved some randomly generated instances, which form also the benchmark for the algorithms by Billionnet and Calmels [2] and Michelon and Veilleux [13].

The randomly generated instances by Gallo, Hammer and Simeone are constructed as follows. Let $\Delta$ 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 $\Delta$, 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 the following tables we consider instances with densities $\Delta = 25\%, 50\%, 75\%$ and $100\%$ and with $n$ up to 1500. For each size and density we have calculated 10 instances. The entries in the table are average values out of the instances, that could be solved within the time limit of 12 hours. For each density, we report results for $n$ up to 1500 in multiple of 100. All tests where conducted using gcc 3.2.2 on an Intel Pentium4 processor at 2.4 GHz. Table 1 reports the number of instances solved. For density 50% and above most instances were solved, while for $\Delta = 25$ some unsolved instances remain.

In Table 2, the average computational times are reported. Most instances are solved within a few hours of cpu-time on average, which is reasonable for these large-sized instances. The following Table 3 shows the corresponding time used in the reduction phase. By comparing Table 2 and Table 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 instances contain well below 100 items which is much more manageable for the branch-and-bound algorithm.

Finally, we compare the present algorithm with that of Caprara, Pisinger, Toth 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 are only shown, whenever all 10 instances could be solved within the time limit of 12 hours. 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 instances of large size, and in particular low-density problems can only be solved for slightly more than 100 items. These experiments illustrate that reduction is a key ingredients in solving difficult $\mathcal{NP}$-hard optimization problems.

| $n/\delta$ | 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 |
| 1000 | 6 | 10 | 10 | 10 |
| 1100 | 2 | 9 | 10 | 10 |
| 1200 | 5 | 9 | 10 | 9 |
| 1300 | 1 | 9 | 10 | 8 |
| 1400 | 4 | 10 | 9 | 9 |
| 1500 | 3 | 6 | 10 | 10 |
| avg. | 6.3 | 9.5 | 9.9 | 9.7 |

Table 1: Number of instances solved out of 10

| $n/\delta$ | 25 | 50 | 75 | 100 |
|---|---|---|---|---|
| 100 | 210.7 | 54.2 | 6.7 | 2.7 |
| 200 | 860.0 | 168.9 | 23.0 | 76.5 |
| 300 | 4031.9 | 556.8 | 94.7 | 90.3 |
| 400 | 1190.1 | 978.3 | 295.0 | 173.2 |
| 500 | 3865.7 | 1982.8 | 266.6 | 392.2 |
| 600 | 5565.4 | 3711.1 | 744.5 | 794.7 |
| 700 | 10422.0 | 3158.7 | 1471.6 | 575.5 |
| 800 | 2916.5 | 2939.4 | 1768.4 | 1375.2 |
| 900 | 20440.0 | 2461.9 | 2585.2 | 971.7 |
| 1000 | 13281.9 | 8336.6 | 2419.7 | 1809.9 |
| 1100 | 8090.1 | 5692.1 | 3929.6 | 2263.6 |
| 1200 | 446.1 | 11544.4 | 4093.2 | 5002.3 |
| 1300 | 3393.8 | 15948.2 | 6644.5 | 3386.3 |
| 1400 | 6717.0 | 22606.5 | 10947.7 | 5416.3 |
| 1500 | 5077.9 | 19135.8 | 12770.7 | 12771.1 |
| avg. | 5767.3 | 6618.4 | 3204.1 | 2340.1 |

Table 2: Total computation time in seconds (average of solved instances)

| $n/\delta$ | 25 | 50 | 75 | 100 |
|---|---|---|---|---|
| 100 | 189.2 | 54.2 | 6.7 | 2.7 |
| 200 | 584.0 | 168.7 | 23.0 | 76.5 |
| 300 | 3207.8 | 556.7 | 94.7 | 90.1 |
| 400 | 1189.9 | 975.6 | 294.8 | 173.0 |
| 500 | 3864.6 | 1982.7 | 266.6 | 391.9 |
| 600 | 5564.8 | 3611.3 | 743.5 | 794.5 |
| 700 | 10421.7 | 3141.9 | 1470.0 | 575.2 |
| 800 | 2916.3 | 2938.6 | 1764.5 | 1374.6 |
| 900 | 20439.8 | 2461.6 | 2579.2 | 971.3 |
| 1000 | 13281.1 | 8298.9 | 2419.6 | 1740.8 |
| 1100 | 8089.9 | 5690.1 | 3924.0 | 2246.9 |
| 1200 | 446.0 | 11538.6 | 4076.0 | 4052.8 |
| 1300 | 3393.6 | 15939.7 | 6642.0 | 3385.4 |
| 1400 | 6714.8 | 22512.9 | 8305.5 | 5412.7 |
| 1500 | 4919.9 | 19115.6 | 12755.1 | 12414.4 |
| avg. | 5681.6 | 6599.1 | 3024.3 | 2246.9 |

Table 3: Time used by reduction phase (average of solved instances)

| $n/\delta$ | 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 |
| 1000 | 37.3 | 46.2 | 17.3 | 28.1 |
| 1100 | 39.0 | 24.7 | 43.1 | 28.0 |
| 1200 | 21.2 | 37.1 | 47.6 | 77.6 |
| 1300 | 45.0 | 57.6 | 58.2 | 40.5 |
| 1400 | 58.0 | 125.7 | 58.8 | 28.2 |
| 1500 | 120.7 | 39.2 | 44.6 | 56.8 |
| avg. | 41.6 | 36.9 | 30.6 | 26.4 |

Table 4: Size of instance after reduction phase (average of solved instances)

| $n/\delta$ | 25 | 50 | 75 | 100 |
|---|---|---|---|---|
| 100 | 43631.2 | 91887.9 | 192668.3 | 260186.2 |
| 200 | 193798.8 | 561313.4 | 448942.6 | 887270.1 |
| 300 | 599935.6 | 1093950.3 | 1968163.6 | 2546920.8 |
| 400 | 1408773.4 | 2140089.4 | 3298495.0 | 4083908.0 |
| 500 | 1684972.3 | 2987465.9 | 4402497.5 | 7078125.3 |
| 600 | 3488900.1 | 4084686.4 | 8384937.0 | 8475661.0 |
| 700 | 2661969.2 | 7588377.9 | 8654936.4 | 11320611.2 |
| 800 | 3866772.2 | 6219035.0 | 11137681.6 | 19685467.6 |
| 900 | 5155890.5 | 9998054.0 | 9526540.9 | 16137230.3 |
| 1000 | 8886079.3 | 14464274.4 | 17338565.3 | 25372726.9 |
| 1100 | 3644757.5 | 15561306.1 | 22334441.4 | 43524304.8 |
| 1200 | 15216608.0 | 19180365.4 | 32351394.2 | 40863251.4 |
| 1300 | 1389218.0 | 18301520.0 | 34343179.1 | 27058156.2 |
| 1400 | 21651174.5 | 26857966.2 | 34111466.4 | 58407674.9 |
| 1500 | 24252955.0 | 23208258.3 | 42625118.1 | 37499235.4 |
| avg. | 6276362.4 | 10155903.4 | 15407935.2 | 20213382.0 |

Table 5: Objective value (average of solved instances)

| $n/\delta$ | 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 |

Table 6: Total computation time in seconds, Caprara, Pisinger, Toth (average of 10 instances)

## 7 Conclusions

The QKP is an important model with numerous applications. Due to its intrinsic difficulty it has attained much interest in the last decade. Much of the research is of method developing nature in which the goal is to develop general techniques which can be applied to a large variety of related problems.

The QKP problem is a difficult $\mathcal{NP}$-hard optimization problem, in the sense that existing branch-and-bound methods are not able to solve instances of large size. For such problems, reduction plays a key ingredients 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 a great potential. In many instances, aggressive reduction reduces the instance to a trivial or at least manageable size.

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

From a practical point of view, the developed algorithm is able to solve instances much larger than previously reported in the literature. Previous approaches [4] have solved dense instances with up to 400 variables. This limit has now been pushed forward to instances with up to 1500 variables. An even more important achievement is, that the new techniques also work for medium and low-density instances, where previous techniques mainly worked for high-density instances.

## 8 Acknowledgements

We would like to thank the August Krogh Institute and Zoological Museum, University of Copenhagen for having made the *BioCluster* supercomputer available for these experiments.

## References

[1] E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28:1130–1154, 1980.

[2] A. Billionnet and F. Calmels. Linear programming for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 92:310–325, 1996.

[3] A. Billionnet, A. Faye, and E. Soutif. A new upper-bound and an exact algorithm for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 112:664–672, 1999.

[4] A. Caprara, D. Pisinger, and P. Toth. Exact solution of the quadratic knapsack problem. *INFORMS Journal on Computing*, 11:125–137, 1999.

[5] P. Chaillou, P. Hansen, and Y. Mahieu. Best network flow bound for the quadratic knapsack problem. In B. Simeone, editor, *Combinatorial Optimization*, volume 1403 of *lecture notes in mathematics*, pages 225–235. Springer, 1989.

[6] W. Cook and P. Seymour. Tour mergining via branch-decomposition. 2002. http://www.isye.gatech.edu/ wcook/papers/tmerge.ps.

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

[8] G. Gallo, P.L. Hammer, and B. Simeone. Quadratic knapsack problems. *Mathematical Programming Study*, 12:132–149, 1980. This is the correct journal.

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

[10] C. Helmberg, F. Rendl, and R. Weismantel. Quadratic knapsack relaxations using cutting planes and semidefinite programming. In W.H. Cunningham, S.T. McCormick, and M. Queyranne, editors, *Proceedings of the Fifth IPCO Conference*, volume 1084, pages 175–189. Springer Verlag, 1996.

[11] E.L. Johnson, A. Mehrotra, and G.L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62:133–152, 1993.

[12] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2003.

[13] P. Michelon and L. Veilleux. Lagrangean methods for the 0-1 quadratic knapsack problem. *European Journal of Operational Research*, 92:326–341, 1996.

[14] K. Park, K. Lee, and S. Park. An extended formulation approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, 95:671–682, 1996.

[15] T. Polzin and S. Vahdati. Extending reduction techniques for the steiner tree problem: A combination of alternative-and bound-based approaches. Technical Report MPI-I-2001-1-007, Max-Planck-Institut für Informatik, 2001.

[16] A.B. Rasmussen and R. Sandvik. Kvaliteten af grænseværdier for det kvadratiske knapsack problem, 2002. Project 02-09-7, DIKU, University of Copenhagen (D. Pisinger, supervisor).

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

[18] C. Witzgall. Matematical methods of site selection for electronic message systems (ems). Technical report, NBS internal report, 1975.