INFORMS Journal on Computing

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

A Dynamic Programming Heuristic for the Quadratic Knapsack Problem

Franklin Djeumou Fomeni, Adam N. Letchford

INFORMS Journal on Computing
Vol. 26, No. 1, Winter 2014, pp. 173–182
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.2013.0555

A Dynamic Programming Heuristic for the Quadratic Knapsack Problem

Franklin Djeumou Fomeni, Adam N. Letchford
Department of Management Science, Lancaster University, Lancaster LA1 4YX, United Kingdom
{f.djeumoufomeni@lancaster.ac.uk, a.n.letchford@lancaster.ac.uk}

It is well known that the standard (linear) knapsack problem can be solved exactly by dynamic programming in $\mathcal{O}(nc)$ time, where n is the number of items and c is the capacity of the knapsack. The *quadratic* knapsack problem, on the other hand, is *NP*-hard in the strong sense, which makes it unlikely that it can be solved in pseudo-polynomial time. We show, however, that the dynamic programming approach to the linear knapsack problem can be modified to yield a highly effective constructive heuristic for the quadratic version. In our experiments, the lower bounds obtained by our heuristic were consistently within a fraction of a percent of optimal. Moreover, the addition of a simple local search step enabled us to obtain the optimal solution of all instances considered.

Key words: knapsack problems; integer programming; dynamic programming
History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms; received March 2012; revised September 2012, January 2013; accepted February 2013. Published online in *Articles in Advance* July 22, 2013.

1. Introduction

The *quadratic knapsack problem* (QKP), introduced by Gallo et al. (1980), is an optimisation problem of the following form:

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

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \le c, \tag{2}$$

$$x \in \{0, 1\}^n. \tag{3}$$

Here, n is the number of *items*, c is the (*positive and integral*) *capacity* of the knapsack, w_i are the (positive and integral) *weights* of the items, and q_{ij} are the (nonnegative and integral) *profits*. For $i = 1, \ldots, n$, the binary variable x_i takes the value 1 if and only if the ith item is inserted into the knapsack.

Note that, because $x_i^2 = x_i$ for all i, one can easily incorporate linear terms by adjusting the q_{ii}. Indeed, the QKP reduces to the standard (linear) knapsack problem (KP) when $q_{ij} = 0$ for all $i \ne j$. In terms of computational complexity, however, the QKP is more complex than the KP. Indeed, although the KP is NP-hard, as shown by Karp (1972), it is possible to solve it in $\mathcal{O}(nc)$ time using dynamic programming (DP), as shown by Bellman (1957). The QKP, on the other hand, is known to be NP-hard in the *strong sense* (see, e.g., Caprara et al. 1999), which makes it unlikely that a pseudo-polynomial time algorithm exists. In fact, there is evidence that, even in the special case where $w_i = 1$ for all i, one cannot even *approximate* the optimal profit within a constant factor in polynomial time (e.g., Khot 2006). Even for this special case, the best approximation factor obtained to date is $\mathcal{O}(n^{1/4})$ (Bhaskara et al. 2010).

The QKP is a well-studied combinatorial optimisation problem with a variety of important applications, for example in the location of satellites, airports, railway stations, or freight terminals. A variety of bounding procedures, heuristics, and exact algorithms are available for it. For details, we refer the reader to Kellerer et al. (2004, Ch. 12), and the more recent survey by Pisinger (2007). One of the most effective exact algorithms for the QKP is that of Caprara et al. (1999), which is based on Lagrangian relaxation, subgradient optimisation, and branch and bound. Recently, this algorithm was made even more effective by Pisinger et al. (2007), using several powerful reduction techniques.

The purpose of this paper is to show that the DP approach to the KP can be modified to yield an effective *heuristic* for the QKP. In the basic form of our heuristic, one simply uses the same stages and states as in the standard DP recursion for the KP, but uses a quadratic objective in place of a linear objective. This heuristic already performs surprisingly well, but it can be significantly enhanced with little additional effort. Specifically, we found that if the items are ordered appropriately, and ties are broken in an intelligent way, then one can consistently obtain lower bounds that are within 0.05% of optimality.

Moreover, when we applied a simple local search step to the solutions generated by the enhanced version of our heuristic, we obtained the optimal solution of all instances considered.

The paper is structured as follows: In §2 we review the relevant literature on the KP and QKP. In §3, the new heuristic is described, and it is shown how to implement it to use $\mathcal{O}(n^2c)$ time and $\mathcal{O}(nc)$ memory space. In §4 we present the abovementioned ordering and tie-breaking rules, and show that they considerably enhance the performance of the heuristic. In §5 we present extensive computational results obtained with the enhanced version of our heuristic, along with ones obtained with the additional local search step. Finally, some concluding remarks are presented in §6.

2. Literature Review

Because the literature on knapsack problems is vast, we do not attempt to cover it all here. Instead, we refer the reader to books by Kellerer et al. (2004), Martello and Toth (1990), and the survey by Pisinger (2007). Nevertheless, there are certain key concepts from the literature that are vital to what follows and are therefore covered in the following four subsections.

2.1. The Continuous Relaxation of the KP

The continuous relaxation of the KP is defined as follows:

$$\max\left\{\sum_{i=1}^{n} q_{ii}x_i:\ w^T x \le c,\ x\in[0,1]^n\right\}.$$

Dantzig (1957) showed that this relaxation can be solved quickly, in $\mathcal{O}(n\log n)$ time, as follows:

1. Temporarily set x_i to 0 for all i.
2. Sort the items in nonincreasing order of *profit-to-weight ratio* q_{ii}/w_i.
3. Iteratively set variables to one in the given order, as long as this can be done while maintaining feasibility.
4. Set the next variable to the largest possible (typically fractional) value that it can take, while maintaining feasibility.

In Balas and Zemel (1980) it is shown that, in fact, no sorting is necessary, and one can solve the continuous relaxation of the KP in $\mathcal{O}(n)$ time, using advanced median-finding techniques.

2.2. The Classical Dynamic Programming Approach to the KP

Next, we recall the classical DP approach to the KP. For any $k\in\{1,\dots,n\}$ and $r\in\{0,\dots,c\}$, let $f(k,r)$ be the maximum profit obtainable by packing a selection of the first k items whose total weight is equal to r. That is, let

$$f(k,r) = \max\left\{\sum_{i=1}^{k} p_i x_i:\ \sum_{i=1}^{k} w_i x_i = r,\ x_i\in\{0,1\}\ (i=1,\dots,k)\right\},$$

where $p_i = q_{ii}$.

If no such packing exists, then let $f(k,r)=-\infty$. Also, by convention, let $f(0,0)=0$. Now observe that

$$f(k,r)=\begin{cases}\max\{f(k-1,r),\ f(k-1,r-w_k)+p_k\} & \text{if } r\ge w_k,\\ f(k-1,r) & \text{otherwise.}\end{cases} \tag{4}$$

This is a classic dynamic programming recursive function, which indicates that the KP obeys the so-called "principle of optimality" of Bellman (1957). It enables us to compute the $f(k,r)$ using Algorithm 1.

Algorithm 1 (The classical DP algorithm)

Initialise $f(0,0)$ to 0 and $f(k,r)=-\infty$ for all other k, r.
for $k=1,\dots,n$ **do**
  for $r=0,\dots,c$ **do**
    if $f(k-1,r)>f(k,r)$ **then**
      Set $f(k,r):=f(k-1,r)$.
    end if
    if $r+w_k\le c$ and $f(k-1,r)+p_k>f(k,r+w_k)$ **then**
      Set $f(k,r+w_k):=f(k-1,r)+p_k$.
    end if
  end for
end for
Output $\max_{0\le r\le c} f(n,r)$.

This algorithm clearly runs in $\mathcal{O}(nc)$ time. Note that it outputs only the optimal profit. If you want the optimal solution itself, you have to "trace back" the path from the optimal state to the initial state $f(0,0)$. The time taken to do this is negligible.

We remark that some problems that are intermediate in generality between the KP and the QKP can also be solved exactly in pseudo-polynomial time via dynamic programming. Examples include the *edge series-parallel* QKP (Rader and Woeginger 2002) and *symmetric* QKP (Kellerer and Strusevich 2010).

2.3. Upper Planes for the QKP

Now we move on to the QKP itself.

Gallo et al. (1980) say that a vector $\pi\in\mathbb{Q}_+^n$ is an *upper plane* if the following holds:

$$\pi^T x \ge \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}x_i x_j,\quad \forall x\in\{0,1\}^n.$$

Given an upper plane π, one can compute an upper bound for the QKP by solving the following (linear) KP:

$$\max\{\pi^T x \colon w^T x \le c,\ x \in \{0,1\}^n\}. \tag{5}$$

Gallo et al. (1980) proposed four different upper planes, $\pi^1, \ldots, \pi^4$. These are defined by setting, for all i,

$$\pi_i^1 = \sum_{j=1}^{n} q_{ij},$$

$$\pi_i^2 = q_{ii} + \max\left\{\sum_{j\neq i} q_{ij}x_j \colon \sum_{j=1}^{n} x_j \le m,\ x \in \{0,1\}^n\right\},$$

$$\pi_i^3 = q_{ii} + \max\left\{\sum_{j\neq i} q_{ij}x_j \colon w^T x \le c,\ x \in [0,1]^n\right\},$$

$$\pi_i^4 = q_{ii} + \max\left\{\sum_{j\neq i} q_{ij}x_j \colon w^T x \le c,\ x \in \{0,1\}^n\right\}.$$

The quantity m appearing in the definition of π_i^2 is the maximum number of items that can be fitted into the knapsack. This can be computed by sorting the items in nondecreasing order of their weights, and then inserting the items into the knapsack in the given order, until no more can be inserted.

Clearly, for any i, we have $\pi_i^1 \ge \pi_i^2 \ge \pi_i^4$ and $\pi_i^1 \ge \pi_i^3 \ge \pi_i^4$. On the other hand, for any i, the quantities π_i^1 and π_i^2 can be computed easily in $\mathcal{O}(n)$ and $\mathcal{O}(n \log n)$ time, respectively, whereas computing π_i^4 amounts to solving a (linear) KP, could be time consuming. As for π_i^3, it is possible to compute it in $\mathcal{O}(n)$ time as well, using the median-finding technique of Balas and Zemel (1980), mentioned in §2.1.

According to Gallo et al. (1980), using π_i^3 gives the best trade-off between bound strength and computing time.

2.4. Improved Upper Planes for the QKP

Caprara et al. (1999) noted that one can slightly improve the upper planes π_i^2, π_i^3, and π_i^4 by replacing them with the following:

$$\tilde{\pi}_i^2 = q_{ii} + \max\left\{\sum_{j\neq i} q_{ij}x_j \colon \sum_{j\neq i} x_j \le m-1,\ x_j \in \{0,1\}\ (j\neq i)\right\},$$

$$\tilde{\pi}_i^3 = q_{ii} + \max\left\{\sum_{j\neq i} q_{ij}x_j \colon \sum_{j\neq i} w_j x_j \le c - w_i,\ 0 \le x_j \le 1\ (j\neq i)\right\},$$

$$\tilde{\pi}_i^4 = q_{ii} + \max\left\{\sum_{j\neq i} q_{ij}x_j \colon \sum_{j\neq i} w_j x_j \le c - w_i,\ x_j \in \{0,1\}\ (j\neq i)\right\}.$$

They also show that one can obtain even stronger bounds using either linear programming or Lagrangian relaxation. These upper bounds, along with others based on linear programming, Lagrangian decomposition, and semidefinite programming, are surveyed and compared in Kellerer et al. (2004) and Pisinger (2007).

2.5. Heuristics for the QKP

We close this section by mentioning some heuristics that have been proposed for the QKP. Gallo et al. (1980) noted that, for any given upper plane π, the solution to the KP (5) is also feasible for the QKP, and therefore yields a lower bound. They also proposed to improve such solutions using a simple local search procedure called "fill-up-and-exchange." In each iteration of this procedure, either one item is added to the knapsack, or one item in the knapsack is exchanged for one item outside.

Chaillou et al. (1983) presented a greedy constructive heuristic, in which all items are initially placed in the knapsack, and then items are iteratively removed until feasibility is achieved. The criterion for deciding which item to remove in a given iteration is as follows: For each item i currently in the knapsack, let δ_i be the decrease in the profit that would be incurred if item i were removed. The item to be removed is the one that minimises the "loss-to-weight" ratio δ_i/w_i.

Billionet and Calmels (1996) produced a hybrid method, where the method of Chaillou et al. (1983) is used to form an initial solution, and then the fill-up-and-exchange procedure of Gallo et al. (1980) is used to improve that solution. Their computational results demonstrate that this is a highly effective approach.

Some other, more complex heuristics have been proposed, for example based on Lagrangian relaxation (Caprara et al. 1999) and tabu search (Glover and Kochenberger 2002). For details, we refer the reader to Pisinger (2007).

3. Description of the DP Heuristic

Consider what will happen if we attempt to adapt the classical dynamic programming algorithm (see Algorithm 1) to the QKP. We can define $f(k, r)$ in a natural way, as

$$\max\left\{\sum_{i=1}^{k}\sum_{j=1}^{k} q_{ij}x_i x_j \colon \sum_{i=1}^{k} w_i x_i = r,\ x_i \in \{0,1\}\ (i = 1, \ldots, k)\right\}.$$

The problem is that there is no analogue of the recursive Equation (4). Indeed, the Bellman principle of optimality (Bellman 1957) does not hold for the QKP. The following simple example makes this clear.

Example. Suppose that $n = 3$, $c = 2$, and $w_1 = w_2 = w_3 = 1$, and suppose that the profit matrix is

$$Q = \begin{pmatrix} 10 & 0 & 0 \\ 0 & 1 & 10 \\ 0 & 10 & 1 \end{pmatrix}.$$

We have $f(2, 1) = 10$, which corresponds to inserting item 1 into the knapsack, and we have $f(2, 2) = 11$, which corresponds to inserting items 1 and 2. On the other hand, we have $f(3, 2) = 22$, which corresponds to inserting items 2 and 3. Note that item 1 is inserted in the first two cases, but not in the third case. There is therefore no recursive formula that can express $f(3, 2)$ in terms of $f(2, 1)$ and $f(2, 2)$. (This is of course to be expected, given that the QKP is NP-hard in the strong sense.)

The main point of this paper, however, is to show that one can devise an effective heuristic using dynamic programming. To do this, it is helpful to redefine $f(k, r)$ as the *profit of the best packing found by the heuristic* that uses a selection of the first k items and whose total weight is equal to r. It is also helpful to define $S(k, r)$ as the set of items (viewed as a subset of $\{1, \ldots, k\}$) that gives the profit $f(k, r)$. Our heuristic then performs as described in Algorithm 2.

Algorithm 2 (Dynamic programming for the QKP)

Initialise $f(0, 0)$ to 0, and $f(k, r)$ to $-\infty$
  for all other k, r.
Initialise $S(k, r) = \varnothing$ for all k, r.
for $k = 1, \ldots, n$ **do**
  for $r = 0, \ldots, c$ **do**
    if $f(k-1, r) > f(k, r)$ **then**
      Set $f(k, r) := f(k-1, r)$ and
        $S(k, r) := S(k-1, r)$.
    end if
    if $r + w_k \le c$ **then**
      Let β be the profit of $S(k-1, r) \cup \{k\}$.
      if $\beta > f(k, r + w_k)$ **then**
        Set $f(k, r + w_k) := \beta$,
        Set $S(k, r + w_k) := S(k-1, r) \cup \{k\}$.
      end if
    end if
  end for
end for
Let $r^* = \arg\max_{0 \le r \le c} f(n, r)$.
Output the set $S(n, r^*)$ and the associated
  profit $f(n, r^*)$.

Note that the quantity β used in Algorithm 2 can be computed in linear time, using the identity

$$\beta = f(k-1, r) + q_{kk} + 2 \sum_{i \in S(k-1, r)} q_{ik}.$$

The heuristic therefore runs in $\mathcal{O}(n^2 c)$ time.

Table 1 **Dynamic Programming Computations for the Small Example**

	$k = 1$			$k = 2$			$k = 3$		
	β	$f(1, r)$	$S(1, r)$	β	$f(2, r)$	$S(2, r)$	β	$f(3, r)$	$S(3, r)$
$r = 0$	0	0	$\varnothing$	0	0	$\varnothing$	0	0	$\varnothing$
$r = 1$	10	10	{1}	1	10	{1}	1	10	{1}
$r = 2$	*	$-\infty$	$\varnothing$	11	11	{1, 2}	11	11	{1, 3}

Example (Continued). Applying our heuristic to the small example given earlier, we obtain the results shown in Table 1. In this table, the values $f(k, r)$ and the sets $S(k, r)$ are the ones obtained at the *end* of the iteration, for the given k and r. The asterisk on the lower left is to indicate that β is not computed when $k = 1$ and $r = 2$. The maximum of $f(3, r)$ over all r is 11, so $r^* = 2$ and $S(3, r^*) = \{1, 3\}$. The heuristic finds the solution $\{1, 3\}$, with a profit of 11. On the other hand, the optimal solution is $\{2, 3\}$, with a profit of 22.

A drawback of Algorithm 2 is that it takes $\mathcal{O}(n^2 c)$ space, because, for all k and r, we are storing the set $S(k, r)$. Our preliminary computational experiments indicated that Algorithm 2 can only handle problem instances of moderate size because of this large memory requirement.

It is, however, possible to reduce the memory requirement to $\mathcal{O}(nc)$, without any worsening of the running time, by exploiting the fact that all of the computations that take place in a given stage depend only on the values that were computed in the preceding stage. Effectively, this enables us to drop the index k from the $f(k, r)$ array.

The details are as follows: For $r \in \{0, \ldots, c\}$, let $f(r)$ denote the *current* value of the most profitable packing found so far, regardless of the stage k, having a total weight of r. Also, for $r \in \{0, \ldots, c\}$ and $i = 1, \ldots, n$, let $B(r, i)$ be a Boolean variable taking the value 1 if item i is *currently* in the set of items whose profit is $f(r)$, and 0 otherwise. The heuristic can then be implemented as described in Algorithm 3.

Algorithm 3 (Dynamic programming for the QKP with reduced memory)

Initialise $f(r)$ to 0 for $r = 0, \ldots, c$.
Initialise $B(r, i)$ to 0 for all $r = 0, \ldots, c$
  and $i = 1, \ldots, n$.
for $k = 1, \ldots, n$ **do**
  for $r = c$ to 0 (going down in steps of 1) **do**
    if $r \ge w_k$ **then**
      Let β be the profit of $S(k-1, r - w_k) \cup \{k\}$.
      if $\beta > f(r)$ **then**
        Set $f(r) := \beta$,
        Set $B(r, k) := 1$.
      end if
    end if
  end for
end for

Let $r^* = \arg\max_{0 \le r \le c} f(r)$.
Compute the final set of items $S(n, r^*)$.
(This can be done in $\mathcal{O}(n)$ time using the $B(r^*, i)$ values.)
Output $S(n, r^*)$ and the associated profit $f(r^*)$.

Note that in Algorithm 3, the quantity β can still be computed in linear time, using the identity

$$\beta = f(r - w_k) + q_{kk} + 2\sum_{i=1}^{k-1} B(r - w_k, i) q_{ik}.$$

Therefore, Algorithm 3 also runs in $\mathcal{O}(n^2 c)$ time.

The reason that we iteratively decrease r, instead of increasing it, is that the value $f(r)$ at a given stage k depends on the values of $f(r)$ and $f(r - w_k)$ from the previous stage, but does not depend on the value of $f(r')$ from the previous stage, for any $r' > r$.

4. Enhancements

To gain a feeling for the potential of our DP heuristic, we performed some experiments on randomly generated QKP instances. These instances all had $n = 50$, and were created succeeding the following scheme, which is standard in the literature (Billionet and Calmels 1996, Caprara et al. 1999, Gallo et al. 1980, Michelon and Veilleux 1996, Pisinger et al. 2007). For a given *density* Δ%, each profit q_{ij} is zero with probability $(100 - \Delta)$%, and uniformly distributed between 1 and 100 with probability Δ%. Each weight w_i is uniformly distributed between 1 and 50. The capacity c is uniformly distributed between 50 and $\sum_{i=1}^{n} w_i$.

For each instance, we computed the average percentage gap between the lower bound found by our heuristic and the optimal profit. The average gap over 100 instances was 0.26% for $\Delta = 100$% and 16.74% for $\Delta = 25$%. This is very good for the dense instances, but disappointing for the sparse ones. In the following two sections, two simple enhancements to our heuristic are presented that, in combination, drastically improve the quality of the lower bounds obtained, especially for the sparse instances.

4.1. Ordering the Items

Consider again the small example with $n = 3$, presented in the previous section. Observe that if we had reordered the items so that either item 2 or 3 came first, the DP heuristic would have found the optimal solution. So, it is natural to consider the devising of some intelligent way to order the items. Following the papers of Gallo et al. (1980) and Caprara et al. (1999) mentioned in §§2.3 and 2.4, it is natural to consider sorting the items in nonincreasing order of one of the values $\pi_i^1, \ldots, \pi_i^4$, or one of the values $\tilde{\pi}_i^2, \ldots, \tilde{\pi}_i^4$. This gives seven values in total. Moreover, following the idea behind the greedy algorithm for the linear KP, mentioned in §2.1, it is also natural to consider sorting the items in nonincreasing order of any of these seven values divided by the weight w_i. This gives 14 different possible ordering rules in total.

Table 2 **Average Percentage Gaps for Various Bounds When $\Delta = 100$%**

	π_i^1	$\tilde{\pi}_i^2$	π_i^2	$\tilde{\pi}_i^3$	π_i^3	$\tilde{\pi}_i^4$	π_i^4
UB UP	226.24	34.01	36.50	13.54	15.24	13.13	14.84
LB UP	0.87	0.97	1.02	0.29	0.36	0.32	0.35
LB DP	0.19	0.25	0.25	0.17	0.29	0.17	0.35
LB DP/W	0.05	0.06	0.06	0.05	0.05	0.04	0.04

Accordingly, for each of the 200 random instances mentioned earlier, we computed 14 different upper bounds. Moreover, for comparison purposes, we also computed the corresponding seven upper bounds and seven lower bounds obtained from the upper-planes approach (i.e., by solving the knapsack problem (5)).

Tables 2 and 3 display the results obtained. Table 2 is concerned with dense instances and Table 3 with sparse ones. The rows labelled "UB UP" and "LB UP" give the average percentage gaps for the upper and lower bounds obtained from the upper-planes approach, respectively. The row labelled "LB DP" gives the gaps for the lower bounds obtained from our DP heuristic, when the various different π values are used to sort the items. Finally, the row labelled "LB DP/W" gives the same, when the π values divided by the weights w_i are used to sort the items.

It is clear that the upper bounds from the upper-plane approach are rather poor in all cases. The corresponding lower bounds are surprisingly good, especially for dense instances. For our heuristic, the lower bounds for dense instances are very good, especially when the π values are divided by the weights. On the other hand, the various orderings do not help at all for sparse instances. Moreover, bizarrely, dividing the π values by the weights seems to make things worse. This anomalous behaviour is rectified by the second enhancement, described next.

4.2. Breaking Ties Intelligently

When inspecting the solutions obtained by the various algorithms on the sparse instances, we noticed an interesting phenomenon. The optimal solution to such instances almost always inserted more items into

Table 3 **Average Percentage Gaps for Various Bounds When $\Delta = 25$%**

	π_i^1	$\tilde{\pi}_i^2$	π_i^2	$\tilde{\pi}_i^3$	π_i^3	$\tilde{\pi}_i^4$	π_i^4
UB UP	203.23	52.70	52.84	40.61	40.75	39.70	40.86
LB UP	4.06	4.01	4.01	3.39	3.42	3.33	3.43
LB DP	16.47	14.63	14.63	13.79	14.12	13.56	13.90
LB DP/W	25.54	25.13	25.13	24.42	24.61	24.38	24.51

Table 4 Average Percentage Gaps Obtained When Using the Tie-Breaking Rule

	π_i^1	$\tilde{\pi}_i^2$	π_i^2	$\tilde{\pi}_i^3$	π_i^3	$\tilde{\pi}_i^4$	π_i^4
LB DP ($\Delta = 100\%$)	0.19	0.25	0.25	0.17	0.29	0.17	0.35
LB DP/W ($\Delta = 100\%$)	0.05	0.06	0.06	0.05	0.05	0.04	0.04
LB DP ($\Delta = 25\%$)	1.50	1.61	1.61	1.42	1.72	1.62	1.63
LB DP/W ($\Delta = 25\%$)	0.37	0.41	0.41	0.30	0.30	0.30	0.30

the knapsack than the DP heuristic did. This suggests adjusting the DP heuristic slightly so that it gives a preference for (partial) solutions with more items. More precisely, suppose that, for a given k and r in Algorithm 3, we find that $\beta = f(r)$. Then, if $|S(k-1, r-w_k)| + 1 > |S(k, r)|$, we set $S(k, r) = S(k-1, r-w_k) \cup \{k\}$.

We call this enhancement the *tie-breaking* rule. When this rule is used on its own, without any reordering of the items, the average percentage gap over the same instances previously mentioned was 0.26% for $\Delta = 100\%$ and 3.93% for $\Delta = 25\%$. This represents a substantial improvement for the sparse instances, but has little or no effect on the dense instances.

The really interesting results, however, are obtained when reordering and tie breaking are used in combination. Table 4 shows the results obtained when using this option. By comparing Table 4 with Tables 2 and 3, it is apparent that the addition of tie breaking improves the performance dramatically for the sparse instances, without having any effect on the performance for the dense instances.

All things considered, the best option seems to be to order the items in nonincreasing order of π_i^3/w_i, $\tilde{\pi}_i^3/w_i$, π_i^4/w_i, or $\tilde{\pi}_i^4/w_i$, and to use the tie-breaking rule. This gives an average percentage error of 0.05 or less for the dense instances, and 0.30 or less for the sparse instances.

5. Additional Computational Results

In this section, we present more extensive computational results. In §5.1, we present detailed results for the four best variants of our heuristic. In §5.2, we compare the best variant of our heuristic with some other heuristics.

All routines were coded in the C language, compiled with *gcc* 4.4.1 *of Code::Blocks* 10.05, and run on a Dell desktop, with a 3.10 GHz processor and 4 GB of RAM, under the Windows 7 operating system.

5.1. Detailed Results for the Four Best Variants

We present here results for the four most promising variants mentioned in §4.2, i.e., the ones in which the tie-breaking rule is used and the items are considered in nonincreasing order of π_i^3/w_i, $\tilde{\pi}_i^3/w_i$, π_i^4/w_i, or $\tilde{\pi}_i^4/w_i$. The instances were generated using the scheme described at the start of §4, but this time we generated 20 instances for $n \in \{20, 40, \ldots, 380\}$ and for $\Delta\% \in \{25\%, 50\%, 75\%, 100\%\}$.

The results for the four variants are shown in Tables 5–8. It will be seen that, for each value of $\Delta\%$, the range of n varies. This is because we had to solve each instance to proven optimality to compute the percentage gaps, but this was possible only for certain values of Δ and n. (To solve the instances to optimality, we used the code of Caprara et al. 1999, which was kindly given to us by the late Alberto Caprara.)

Table 5 Average Percentage Gaps When Items Are Sorted According to π_i^3/w_i

Instances		DP heuristic			UP			Improvement
Δ (%)	n	Avg. gap	Std. dev.	Avg. time	Avg. gap	Std. dev.	Avg. time	Perf (%)
100	20	0.002	0.01	0.003	0.30	0.61	0.000	71.20
	40	0.000	0.000	0.007	1.05	1.40	0.000	100.00
	60	0.01	0.03	0.047	0.08	0.13	0.001	78.36
	80	0.03	0.09	0.079	0.08	0.11	0.002	62.33
	100	0.01	0.04	0.176	0.10	0.19	0.005	86.60
	120	0.02	0.05	0.244	0.12	0.18	0.010	83.26
	140	0.04	0.12	0.120	0.15	0.36	0.011	68.26
	160	0.002	0.007	0.805	0.10	0.19	0.016	97.82
	180	0.005	0.01	0.521	0.06	0.07	0.020	91.67
	200	0.002	0.006	0.862	0.05	0.04	0.027	94.66
	220	0.006	0.01	1.090	0.03	0.05	0.035	82.79
	240	0.01	0.03	1.910	0.07	0.13	0.044	74.87
	260	0.04	0.15	1.440	0.09	0.18	0.055	50.73
	280	0.009	0.01	2.793	0.03	0.04	0.069	77.32
	300	0.005	0.01	6.142	0.08	0.14	0.140	93.99
	320	0.006	0.009	4.040	0.05	0.04	0.100	87.85
	340	0.01	0.01	1.268	0.10	0.04	0.114	89.35
	360	0.003	0.005	5.647	0.04	0.04	0.140	93.10
	380	0.004	0.003	6.673	0.009	0.003	0.163	54.38
75	20	0.21	0.42	0.002	0.63	1.21	0.000	66.44
	40	0.03	0.07	0.009	0.32	0.57	0.000	88.70
	60	0.17	0.62	0.019	0.86	1.14	0.001	79.31
	80	0.01	0.03	0.045	0.36	0.38	0.002	95.21
	100	0.01	0.03	0.076	0.30	0.49	0.004	94.29
	120	0.04	0.12	0.135	0.40	0.55	0.006	89.22
	140	0.01	0.02	0.270	0.27	0.53	0.010	96.33
	160	0.02	0.05	0.425	0.21	0.26	0.014	89.22
	180	0.01	0.02	0.510	0.18	0.21	0.020	93.56
	200	0.03	0.05	0.700	0.23	0.28	0.026	86.99
50	20	0.49	1.60	0.002	2.18	4.10	0.000	77.29
	40	0.16	0.36	0.008	1.08	1.77	0.000	84.49
	60	0.09	0.19	0.021	1.17	1.97	0.001	92.03
	80	0.05	0.16	0.046	0.73	1.17	0.002	92.52
	100	0.01	0.02	0.071	0.89	0.81	0.004	98.83
	120	0.06	0.13	0.149	0.74	1.17	0.006	90.90
	140	0.06	0.16	0.203	0.46	0.64	0.009	85.66
	160	0.02	0.07	0.387	0.40	0.27	0.014	92.59
25	20	0.15	0.47	0.002	3.98	5.60	0.000	96.15
	40	0.55	1.51	0.008	4.04	5.03	0.000	86.31
	60	0.24	0.54	0.018	2.91	3.28	0.001	91.68
	80	0.28	0.39	0.043	2.14	2.29	0.002	86.52
	100	0.29	0.39	0.074	2.71	2.81	0.003	89.24
	120	0.19	0.28	0.123	2.12	2.50	0.006	91.02
	140	0.14	0.21	0.231	1.78	1.70	0.009	92.07

Table 6 Average Percentage Gaps When Items Are Sorted According to $\tilde{\pi}_i^3/w_i$

Instances		DP heuristic			UP			Improvement
Δ (%)	n	Avg. gap	Std. dev.	Avg. time	Avg. gap	Std. dev.	Avg. time	Perf (%)
100	20	0.08	0.22	0.001	0.59	0.99	0.000	85.29
	40	0.002	0.01	0.008	0.53	0.69	0.000	99.52
	60	0.01	0.03	0.047	0.08	0.13	0.001	78.36
	80	0.03	0.09	0.080	0.11	0.18	0.002	73.17
	100	0.01	0.04	0.176	0.10	0.19	0.005	86.60
	120	0.02	0.05	0.252	0.12	0.18	0.007	83.60
	140	0.04	0.012	0.464	0.11	0.30	0.011	56.96
	160	0.002	0.007	0.803	0.10	0.19	0.017	97.82
	180	0.005	0.01	0.510	0.06	0.07	0.020	91.67
	200	0.001	0.006	0.899	0.05	0.04	0.027	94.74
	220	0.005	0.007	1.030	0.03	0.05	0.035	85.87
	240	0.02	0.03	1.500	0.08	0.14	0.045	74.50
	260	0.04	0.15	1.533	0.09	0.18	0.055	50.73
	280	0.009	0.01	2.607	0.03	0.04	0.069	77.32
	300	0.005	0.01	6.094	0.08	0.14	0.093	93.99
	320	0.006	0.01	3.364	0.06	0.06	0.099	88.95
	340	0.01	0.01	1.355	0.10	0.04	0.113	89.35
	360	0.004	0.007	5.487	0.04	0.03	0.140	89.93
	380	0.004	0.003	6.316	0.009	0.006	0.163	54.38
75	20	0.21	0.42	0.002	0.63	1.21	0.000	66.44
	40	0.03	0.07	0.008	0.25	0.47	0.000	85.42
	60	0.18	0.62	0.021	0.79	1.03	0.001	77.40
	80	0.01	0.03	0.045	0.36	0.38	0.002	95.49
	100	0.07	0.26	0.076	0.34	0.56	0.004	74.14
	120	0.04	0.12	0.132	0.40	0.55	0.006	89.54
	140	0.01	0.02	0.275	0.27	0.53	0.010	96.33
	160	0.02	0.05	0.420	0.21	0.28	0.015	89.44
	180	0.01	0.02	0.504	0.18	0.21	0.019	92.21
	200	0.03	0.05	0.640	0.23	0.28	0.026	86.93
50	20	0.55	1.57	0.002	1.68	3.36	0.000	67.20
	40	0.09	0.18	0.008	1.20	1.90	0.000	92.34
	60	0.09	0.19	0.021	1.39	2.05	0.001	93.31
	80	0.05	0.16	0.046	0.73	1.17	0.002	92.49
	100	0.01	0.02	0.072	1.04	1.33	0.003	99.99
	120	0.06	0.13	0.149	0.72	1.16	0.006	91.65
	140	0.06	0.16	0.200	0.46	0.64	0.009	85.66
	160	0.02	0.07	0.380	0.38	0.27	0.014	92.36
25	20	0.15	0.47	0.002	3.54	4.47	0.000	95.68
	40	0.55	1.51	0.008	3.86	5.11	0.000	85.65
	60	0.21	0.53	0.017	2.81	3.21	0.001	92.32
	80	0.28	0.39	0.045	2.69	4.13	0.002	89.28
	100	0.29	0.39	0.074	2.74	2.81	0.003	89.32
	120	0.19	0.27	0.126	2.00	2.48	0.006	90.41
	140	0.14	0.21	0.227	1.78	1.70	0.009	92.07

Table 7 Average Percentage Gaps When Items Are Sorted According to π_i^4/w_i

Instances		DP heuristic			UP			Improvement
Δ (%)	n	Avg. gap	Std. dev.	Avg. time	Avg. gap	Std. dev.	Avg. time	Perf (%)
100	20	0.08	0.22	0.004	0.43	0.81	0.000	79.76
	40	0.000	0.000	0.012	0.61	0.79	0.005	100.00
	60	0.01	0.04	0.093	0.07	0.12	0.048	76.22
	80	0.03	0.09	0.159	0.08	0.12	0.082	64.53
	100	0.01	0.04	0.351	0.10	0.19	0.179	86.60
	120	0.02	0.05	0.488	0.12	0.18	0.253	82.14
	140	0.04	0.12	0.910	0.11	0.30	0.445	56.25
	160	0.002	0.007	1.568	0.10	0.19	0.742	97.82
	180	0.005	0.07	0.996	0.07	0.49	0.496	92.03
	200	0.002	0.006	1.666	0.05	0.04	0.810	95.66
	220	0.006	0.01	2.000	0.03	0.05	0.953	82.41
	240	0.01	0.03	3.160	0.08	0.14	1.297	78.84
	260	0.04	0.15	2.775	0.09	0.18	1.298	50.35
	280	0.009	0.01	4.822	0.03	0.04	1.988	77.32
	300	0.005	0.01	10.96	0.08	0.14	6.065	93.99
	320	0.005	0.009	6.498	0.05	0.04	2.475	90.07
	340	0.01	0.01	2.320	0.10	0.04	1.216	88.51
	360	0.003	0.005	9.199	0.04	0.04	3.549	93.03
	380	0.004	0.003	10.71	0.009	0.003	4.191	54.38
75	20	0.21	0.42	0.003	0.63	1.21	0.000	66.44
	40	0.03	0.07	0.014	0.33	0.60	0.006	89.10
	60	0.15	0.48	0.040	0.83	1.14	0.019	80.86
	80	0.02	0.04	0.087	0.36	0.38	0.045	94.36
	100	0.07	0.26	0.150	0.32	0.51	0.077	76.83
	120	0.03	0.12	0.259	0.34	0.41	0.132	88.67
	140	0.01	0.02	0.538	0.27	0.53	0.271	96.21
	160	0.01	0.05	0.830	0.21	0.26	0.411	91.04
	180	0.01	0.02	1.004	0.18	0.21	0.511	93.56
	200	0.02	0.05	1.296	0.23	0.28	0.610	90.45
50	20	0.48	1.56	0.003	2.37	4.08	0.001	79.48
	40	0.16	0.36	0.015	0.89	1.50	0.007	81.15
	60	0.11	0.20	0.040	1.17	1.97	0.020	89.84
	80	0.05	0.16	0.089	0.73	1.17	0.046	92.48
	100	0.01	0.03	0.144	1.04	1.33	0.073	98.19
	120	0.06	0.13	0.287	0.74	1.17	0.146	90.90
	140	0.06	0.16	0.167	0.46	0.64	0.199	85.45
	160	0.02	0.07	0.766	0.40	0.27	0.375	92.95
25	20	0.05	0.25	0.003	3.98	5.60	0.000	98.54
	40	0.55	1.52	0.013	4.04	5.03	0.006	86.17
	60	0.22	0.53	0.032	2.91	3.28	0.016	91.26
	80	0.28	0.39	0.083	2.14	2.29	0.042	86.69
	100	0.31	0.40	0.074	2.43	2.32	0.071	86.88
	120	0.19	0.28	0.240	2.04	2.22	0.121	90.53
	140	0.14	0.21	0.457	1.78	1.70	0.230	92.07

For each chosen pair of Δ and n, the tables give the average percentage gap from the optimum, the standard deviation of this gap, and the time taken to compute the bound (in seconds), for both the DP heuristic and the upper-planes heuristic. Also, the last column displays what we call the "performance" of the DP heuristic, which is defined as

$$\frac{\text{Avg. Gap from UP} - \text{Avg. Gap from DP}}{\text{Avg. Gap from UP}} \times 100\%.$$

These results clearly demonstrate the excellent quality of the DP heuristic, because the average gap is a fraction of 1% in all cases. They also demonstrate the superiority of the DP heuristic over the upper-planes heuristic, because the "performance" is almost always well above 80%. Moreover, the running times are quite reasonable, despite the fact that the heuristic runs in pseudo-polynomial time, rather than polynomial time.

Observe that the running times are slightly larger in Tables 7 and 8 than in Tables 5 and 6. This is because 0-1 KP instances must be solved to compute the upper planes π^4 and $\tilde{\pi}^4$, whereas only the continuous relaxations must be solved to compute π^3

Table 8 Average Percentage Gaps When Items Are Sorted According to $\tilde{\pi}_i^4/w_i$

Instances		DP heuristic			UP			Improvement
Δ (%)	n	Avg. gap	Std. dev.	Avg. time	Avg. gap	Std. dev.	Avg. time	Perf (%)
100	20	0.11	0.26	0.005	1.00	1.24	0.001	89.00
	40	0.000	0.000	0.010	0.46	0.49	0.005	100.00
	60	0.01	0.04	0.093	0.03	0.09	0.047	66.66
	80	0.04	0.09	0.156	0.07	0.09	0.080	42.85
	100	0.02	0.05	0.217	0.09	0.08	0.106	77.77
	120	0.01	0.01	0.496	0.07	0.05	0.256	85.71
	140	0.006	0.009	0.896	0.02	0.03	0.436	70.00
	160	0.001	0.003	1.502	0.17	0.25	0.738	99.41
	180	0.004	0.007	0.965	0.08	0.09	0.483	95.00
	200	0.001	0.003	1.687	0.04	0.03	0.793	97.50
	220	0.008	0.01	1.907	0.04	0.06	0.932	80.00
	240	0.02	0.03	2.879	0.11	0.19	1.273	81.81
	260	0.02	0.20	2.772	0.13	0.24	1.268	84.61
	280	0.008	0.01	4.907	0.05	0.05	1.964	84.00
	300	0.001	0.02	12.05	0.04	0.03	5.334	97.50
	320	0.007	0.01	4.245	0.04	0.06	1.881	82.50
	340	0.01	0.01	2.302	0.11	0.04	1.182	90.90
	360	0.004	0.007	7.095	0.04	0.04	3.121	90.00
	380	0.004	0.003	9.982	0.09	0.06	3.953	93.03
75	20	0.29	0.53	0.005	1.04	1.65	0.002	72.11
	40	0.04	0.07	0.014	0.20	0.28	0.006	80.00
	60	0.06	0.09	0.039	0.60	0.90	0.019	90.00
	80	0.02	0.05	0.087	0.29	0.31	0.044	93.10
	100	0.02	0.04	0.149	0.21	0.25	0.075	90.47
	120	0.008	0.01	0.255	0.41	0.52	0.129	98.04
	140	0.01	0.02	0.523	0.34	0.62	0.264	97.05
	160	0.01	0.02	0.826	0.15	0.13	0.405	93.33
	180	0.01	0.02	1.020	0.09	0.12	0.499	88.88
	200	0.03	0.05	1.241	0.17	0.16	0.598	82.35
50	20	0.29	0.41	0.003	2.48	3.93	0.001	88.30
	40	0.07	0.16	0.015	0.84	1.33	0.007	91.60
	60	0.09	0.21	0.040	2.79	0.68	0.020	96.77
	80	0.01	0.03	0.089	1.12	1.42	0.045	99.10
	100	0.02	0.03	0.139	1.51	0.70	0.071	98.67
	120	0.04	0.07	0.285	0.77	1.46	0.144	94.80
	140	0.01	0.02	0.381	0.24	0.16	0.194	95.83
	160	0.007	0.01	0.729	0.35	0.28	0.370	98.00
25	20	0.11	2.26	0.003	3.73	4.69	0.000	97.05
	40	0.99	2.03	0.003	3.81	4.62	0.000	74.01
	60	0.34	0.71	0.032	3.17	3.97	0.000	89.27
	80	0.34	0.40	0.083	2.89	2.88	0.040	88.23
	100	0.22	0.36	0.140	3.01	3.41	0.069	92.69
	120	0.29	0.30	0.236	2.68	2.60	0.118	89.17
	140	0.14	0.21	0.444	1.78	1.70	0.224	92.13

and $\tilde{\pi}^3$. On the other hand, there does not appear to be any significant difference between the percentage gaps obtained by the four variants of the heuristic.

We remark that, whereas the exact algorithm could not be applied to larger instances without running into time and memory difficulties, the DP heuristic can be applied to much larger instances without any difficulty at all, provided that the knapsack capacity c is not excessively large.

5.2. Comparison with Other Heuristics

In this section, we compare our heuristic with the heuristics of Chaillou et al. (1983) (CHM) and Billionet and Calmels (1996) (BC) described in §2.5. We also examine the effect of applying the fill-up-and-exchange procedure of Gallo et al. (1980), also described in §2.5, to the solutions generated by our heuristic. For brevity, we consider only one ordering rule: the one given by $\tilde{\pi}_i^3/w_i$.

Table 9 presents, for each pair Δ and n, the average percentage gap and average running time of four heuristics. The first is the abovementioned variant of our heuristic. (The results in the corresponding two columns are taken from Table 6.) The second is the same variant of our heuristic, followed by the fill-up-and-exchange (FE) procedure. The third is the heuristic of Chaillou et al. (1983), and the fourth is that of Billionet and Calmels (1996). (Recall that the fourth heuristic consists of the third heuristic, followed by FE.)

Three things are clear from this table: First, the DP heuristic gives significantly better solutions than the CHM heuristic, but with comparable running times. Second, the addition of the fill-up-and-exchange step leads to significant improvements in the solutions, though at the cost of additional running time. Third, and most surprising of all, the addition of the fill-up-and-exchange step to the DP heuristic leads to the optimal solution in all cases considered.

6. Concluding Remarks

We have shown that the classical dynamic programming approach to the standard (linear) knapsack problem can be modified to yield an effective heuristic for the quadratic version. Specifically, our computational results demonstrate that, provided the items are ordered correctly and the tie-breaking rule is incorporated, one can consistently compute lower bounds that are within 0.05% of optimal. We are not aware of any other constructive heuristic for the QKP that is capable of consistently finding solutions of such high quality. This high performance is especially remarkable in light of the inapproximability result of Khot (2006), mentioned in §1. Another surprising fact is that the addition of a simple local search step to our heuristic led to the optimal solution being found for all instances considered.

On the other hand, because our heuristic runs in $\mathcal{O}(n^2c)$ time and takes $\mathcal{O}(nc)$ space, it could not be used for instances with very large values of c. A possible way to proceed when c is large is to divide the weights w_i and the knapsack capacity c by a constant, round up each weight to the nearest integer, and round down the capacity. This could cause some feasible solutions to be lost, but would make the running time more manageable.

Another possible approach worth exploring is the following: A folklore trick in the literature on the standard KP (see, e.g., §§2.3 and 2.6 of

Table 9 **DP Heuristic with $\tilde{\pi}_i^3/w_i$ Compared to Other Heuristics**

Instance		DP		DP with FE		CHM		BC	
Δ(%)	n	Avg. gap	Avg. time	Avg. gap	Avg. time	Avg. gap	Avg. time	Avg. gap	Avg. time
100	20	0.08	0.001	0.000	0.006	4.59	0.000	2.03	0.000
	40	0.002	0.008	0.000	0.015	2.37	0.002	0.46	0.003
	60	0.01	0.047	0.000	0.157	2.13	0.016	0.63	0.010
	80	0.03	0.080	0.000	0.166	1.19	0.042	0.22	0.043
	100	0.01	0.176	0.000	0.267	1.26	0.061	0.34	0.092
	120	0.02	0.252	0.000	0.583	0.57	0.146	0.12	0.228
	140	0.04	0.464	0.000	1.221	1.32	0.228	0.12	0.347
	160	0.002	0.803	0.000	1.879	1.11	0.241	0.54	0.524
	180	0.005	0.510	0.000	1.561	0.63	0.565	0.10	1.091
	200	0.001	0.899	0.000	2.325	0.61	0.778	0.08	1.406
	220	0.005	1.030	0.000	3.017	0.57	1.567	0.16	2.165
	240	0.02	1.500	0.000	4.396	1.29	1.896	0.54	2.542
	260	0.04	1.533	0.000	4.086	0.69	2.532	0.26	4.557
	280	0.009	2.607	0.000	7.797	0.39	2.812	0.09	5.147
	300	0.005	6.094	0.000	14.01	0.81	4.641	0.31	6.692
	320	0.006	3.364	0.000	13.76	0.49	2.235	0.14	11.07
	340	0.01	1.355	0.000	9.721	0.37	9.402	0.07	16.83
	360	0.004	5.487	0.000	12.44	0.30	14.28	0.05	19.84
	380	0.004	6.316	0.000	16.11	0.14	9.322	0.03	15.57
75	20	0.21	0.002	0.000	0.007	7.58	0.000	2.94	0.000
	40	0.03	0.008	0.000	0.018	4.59	0.001	1.69	0.002
	60	0.18	0.021	0.000	0.044	2.80	0.005	0.84	0.014
	80	0.01	0.045	0.000	0.098	2.57	0.033	1.05	0.044
	100	0.07	0.076	0.000	0.178	2.17	0.084	0.62	0.104
	120	0.04	0.132	0.000	0.356	1.27	0.128	0.33	0.227
	140	0.01	0.275	0.000	0.579	0.84	0.291	0.17	0.331
	160	0.02	0.420	0.000	1.015	0.57	0.314	0.06	0.594
	180	0.01	0.504	0.000	1.621	0.65	0.622	0.24	1.123
	200	0.03	0.640	0.000	2.216	0.81	0.922	0.28	1.582
50	20	0.55	0.002	0.000	0.005	4.34	0.000	0.00	0.000
	40	0.09	0.008	0.000	0.017	6.37	0.000	2.29	0.002
	60	0.09	0.021	0.000	0.043	3.54	0.011	1.26	0.013
	80	0.05	0.046	0.000	0.095	4.26	0.029	1.33	0.040
	100	0.01	0.072	0.000	0.185	2.59	0.069	0.52	0.111
	120	0.06	0.149	0.000	0.333	2.43	0.137	0.29	0.189
	140	0.06	0.200	0.000	0.571	3.52	0.268	1.67	0.381
	160	0.02	0.380	0.000	0.991	1.59	0.472	0.38	0.597
25	20	0.15	0.002	0.000	0.003	5.44	0.000	1.41	0.000
	40	0.55	0.008	0.000	0.010	5.60	0.001	0.26	0.003
	60	0.21	0.017	0.000	0.038	2.11	0.009	0.59	0.013
	80	0.28	0.045	0.000	0.085	3.29	0.034	1.72	0.058
	100	0.29	0.074	0.000	0.173	6.78	0.069	1.69	0.103
	120	0.19	0.126	0.000	0.331	4.32	0.106	0.58	0.212
	140	0.14	0.227	0.000	0.604	1.25	0.227	0.58	0.377

Kellerer et al. 2004) is to define the following function:

$$f'(k,r) = \min\left\{\sum_{i=1}^{k} w_i x_i\colon\ \sum_{i=1}^{k} p_i x_i \geq r,\ x_i \in \{0,1\}\ (i=1,\ldots,k)\right\}.$$

Using a modified dynamic programming recursion, one can compute $f'(n,r)$ for $k=1,\ldots,n$ and $r=0,\ldots,p^*$, where p^* is the optimal profit of the KP instance. The running time then becomes $\mathcal{O}(np^*)$ rather than $\mathcal{O}(nc)$. It may be possible to adapt this idea to the QKP, and thereby obtain a DP heuristic that is suitable for instances where the capacity c is large but the profits q_{ij} are small.

Acknowledgments
The authors are grateful to the late Alberto Caprara for kindly providing us with a copy of the exact QKP code described in the paper (Caprara et al. 1999).

References
Balas E, Zemel E (1980) An algorithm for large zero-one knapsack problems. *Oper. Res.* 28:1130–1154.
Bellman RE (1957) *Exact Solution of the Quadratic Knapsack Problem* (Princeton University Press, Princeton, NJ).

Bhaskara A, Charikar M, Chlamtac E, Feige U, Vijayaraghavan A, eds. (2010) Detecting high log-densities: An $\mathcal{O}(n^{1/4})$ approximation for densest k-subgraph. Schulman LJ, ed. *Proc. 42nd ACM Sympos. Theory Comput.* (ACM Press, New York), 201–210.

Billionet A, Calmels F (1996) Linear programming for the quadratic knapsack problem. *Eur. J. Oper. Res.* 92:310–325.

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

Chaillou P, Hansen P, Mahieu Y (1983) Best network flow bound for the quadratic knapsack problem. Simeone B, ed. *Combinatorial Optimization*, Lecture Notes in Mathematics, Vol. 1403 (Springer, Berlin), 225–235.

Dantzig GB (1957) Discrete-variable extremum problems. *Oper. Res.* 5:266–288.

Gallo G, Hammer PL, Simeone B (1980) Quadratic knapsack problems. *Math. Program. Stud.* 12:132–149.

Glover F, Kochenberger G, Alidaee B, Amini M (2002) Solving quadratic knapsack problems by reformulation and tabu search, single constraint case. Pardalos PM, Migdalas A, Burkard RE, eds. *Combinatorial and Global Optimization*, Series on Applied Mathematics, Vol. 14 (World Scientific, Singapore), 111–121.

Karp RM (1972) Reducibility among combinatorial problems. Miller RE, Thatcher JW, eds. *Complexity of Computer Computations* (Plenum, New York), 85–103.

Kellerer H, Pferschy U, Pisinger D (2004) *Knapsack Problems* (Springer, Berlin).

Kellerer H, Strusevich VA (2010) Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. *Algorithmica* 57:769–795.

Khot S (2006) Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM J. Comput.* 36: 1025–1071.

Martello S, Toth P (1990) *Knapsack Problems: Algorithms and Computer Implementations* (John Wiley & Sons, Chichester, UK).

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

Pisinger D (2007) The quadratic knapsack problem—A survey. *Discr. Appl. Math.* 155:623–648.

Pisinger D, Rasmussen AB, Sandvik R (2007) Solution of large quadratic knapsack problems through aggressive reduction. *INFORMS J. Comput.* 19:280–290.

Rader DJ Jr, Woeginger GJ (2002) The quadratic 0-1 knapsack problem with series-parallel support. *Oper. Res. Lett.* 30:159–166.