Les Cahiers du GERAD ISSN: 0711–2440

A lifted-space dynamic programming algorithm for the Quadratic Knapsack Problem

F. Djeumou Fomeni

G–2021–27

May 2021

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2021
– Bibliothèque et Archives Canada, 2021

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2021
– Library and Archives Canada, 2021

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

A lifted-space dynamic programming algorithm for the Quadratic Knapsack Problem

Franklin Djeumou Fomeni [a]

[a] *CIRRELT & Department of Analytics, Operations and Information Technologies (AOTI), UQAM, Montréal (Québec), Canada, H2X 3X2*

djeumou_fomeni.franklin@uqam.ca

May 2021
Les Cahiers du GERAD
G–2021–27

Abstract : The Quadratic Knapsack Problem (QKP) is a well-known combinatorial optimization problem which amounts to maximizing a quadratic function of binary variables, subject to a single linear constraint. It has many applications in finance, logistics, telecommunications, facility location, etc. The QKP is NP-hard in the strong sense and the state-of-the-art algorithm for solving the QKP can only handle problems of small and moderate sizes. In this paper, we present a novel heuristic algorithm for finding good QKP feasible solutions. This algorithm consists of combining the dynamic programming approach with a local search procedure, with the novelty that both are adapted and implemented in the space of lifted variables of the QKP. The algorithm runs in $\mathcal{O}(n^3c)$ times and is able to find optimal solutions to more than 97% of standard instances, about 80% of some well-known hard QKP instances, as well as optimality gaps of 0.1% or less for other instances.

Keywords: Knapsack problems, integer programming, dynamic programming, local search, binary quadratic optimization

Acknowledgements: We gratefully acknowledge the financial support provided by the Natural Sciences and Engineering Council of Canada (NSERC), through its Discovery Grant program. We also gratefully acknowledge the support of the Université du Québec À Montréal through their PARFARC program. Thanks are also due to Calcul Quebec and Compute Canada for providing the author with access to their high-performance computing infrastructure.

1 Introduction

The *Quadratic Knapsack Problem* (QKP) is a much-studied combinatorial optimization problem which amounts to maximizing a quadratic function of binary variables subject to a single knapsack constraint. Given a set of items (with their respective weight and profit measurements) and a knapsack with a limited capacity, the QKP aims at selecting a subset of the items so as to maximize the total profit of the items included in the knapsack. In this case, each item included in the knapsack has an individual profit, while there is additional profit for each pair of items selected too.

Mathematically, the problem can be stated as follows:

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

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

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

here, n is the number of *items*, c is the (positive and integral) *capacity* of the knapsack, the w_i are the (positive and integral) *weights* of the items, and the q_{ij} are the (non-negative 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. Thus for the QKP, not only there is a profit for selecting individual items, but there is also an additional profit for selecting pairs of items, which is reflected in the quadratic term $x_i x_j$. The QKP is a generalization of the standard linear knapsack problem (KP). Indeed, we have $x_i^2 = x_i$ for all i and if we set $q_{ij} = 0$ for all $i \neq j$, the QKP will simply become a KP.

The QKP was first introduced in 1980 by Gallo et al. [15] and has since then attracted a significant amount of attention from researchers in the combinatorial optimization community. It has a wide range of important applications, for example in the location of satellites, airports, railway stations or freight terminals. Unlike the KP that can be solved in pseudo-linear time [3, 19] using dynamic programming (DP), the QKP is known to be $\mathcal{NP}$-hard in the *strong sense* (see, e.g., [9]), which makes it unlikely that a pseudo-polynomial time algorithm exists. In fact, there is evidence that, even in the special case in which $w_i = 1$ for all i, one cannot even *approximate* the optimal profit within a constant factor in polynomial time (e.g., Khot [22]). Even for this special case, the best approximation factor obtained to date is $\mathcal{O}(n^{1/4})$ (Bhaskara et al. [5]).

Existing exact solution methods for the QKP [7, 8, 9, 12] are limited either by the size of the problems that can be solved, or by the amount of time needed to solve these instances. On the other hand, several heuristic and metaheuristic algorithms have been developed for the QKP and have all reported the abilities of producing good quality solutions within a short amount of computational time. These reported results are all based on standard QKP instances that are usually generated following the scheme initially proposed by Gallo et al. [15]. However, in a recent study of the asymptotic behavior of the QKP, Schauer [30] showed that such standard instances may sometimes be relatively easy for modern heuristic algorithms. Additionally, they presented a set of problem instances that can be formulated as QKP and which proved to be very challenging to many existing state-of-the-art QKP heuristic algorithms.

The main contribution in this paper is that we develop a novel deterministic heuristic algorithm for the QKP. This algorithm is based on adapting and implementing the dynamic programming approach in the space of quadratic variables, which can also be referred to as the space of lifted variables. The algorithm is pseudo-polynomial as it runs in $\mathcal{O}(n^3 c)$ times, and uses $\mathcal{O}(nc)$ of memory space. The advantage of moving up in the space of lifted variables is that one can, more efficiently, capture some of the quadratic attributes of the problem, such as the profit contribution of pairs of items, and obtain much better quality solutions than with the existing heuristic solutions. Furthermore, we have also developed a modified version of the 'fill-up-and-exchange' local search procedure in order to improve the DP solutions. This procedure is also designed to operate in the space of lifted variables for much better solutions. We have conducted a thorough computational experiment with a total of

1480 QKP instances, including both standard and challenging QKP instances. The results show that our algorithm can find optimal solutions to more than 97% of the standard QKP instances. It can also find optimal solution to more than 80% of the challenging QKP instances, and for the instances for which optimality cannot be found, the algorithm produces feasible solutions that are within 0.1% (on average) of optimality.

The novelty in our proposed algorithm is the fact that it considers known algorithms such as the DP and the 'fill-up-and-exchange' local search procedure and lifts them in the space of quadratic variables to yield exceptional results, especially for some challenging QKP instances.

The remainder of the paper is as follows. In Section 2, we review the relevant literature on the KP, as well as the existing heuristic and exact solution algorithms for the QKP. In Section 3, our new heuristic is presented along with a complexity analysis, which shows how the algorithm can run in $\mathcal{O}(n^3c)$ times and use $\mathcal{O}(n^3c)$ of memory space. A reduced-memory version that will only require $\mathcal{O}(nc)$ of memory space is then presented in Section 4 along with our local search procedure to improve the Lifted DP solutions. The results of our computational experiments are presented and analyzed in Section 5. Finally, some concluding remarks are made in Section 6.

2 Literature review

Since the literature on knapsack problems is vast, we do not attempt to cover it all here. Instead, we refer the reader to the books [20, 23] and the survey [24]. Nevertheless, there are certain key concepts from the literature which are vital to what follows and are therefore covered in the following four subsections.

2.1 The classical dynamic programming approach to the KP

In this section, we present an implementation of 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 corresponds to:

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

where $p_i = q_{ii}$.

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

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

This is a classic dynamic programming recursive function, which indicates that the KP obeys the so-called Bellman [4] 'principle of optimality'. One can then compute the $f(k,r)$ using Algorithm 1.

This algorithm clearly runs in $\mathcal{O}(nc)$ time, but it only outputs the optimal profit and one then needs to 'trace back' the path from the optimal state to the initial state $f(0,0)$. The time taken to do this is negligible.

2.2 The DP heuristic approach to the QKP

In 2014, Djeumou-Fomeni and Letchford [13] showed that the adaptation of the above classical DP cannot result into an exact solution method for the QKP (except for some problems that are intermediate in generality between the KP and the QKP, e.g., [21, 29]). This is because there is no analogue of the Bellman's principle of optimality in the case of the QKP. They then went on to show, however, that is possible to use the idea of the DP to yield a heuristic algorithm for the QKP. This idea requires to redefine $f(k,r)$ as the profit of the *best packing found* by the heuristic that uses a selection of the

Algorithm 1 : The classical DP algorithm

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

first k items and whose total weight is equal to r. The use of 'best packing' enforces that one has to define a set $S(k,r)$ as the set of items (viewed as a subset of $\{1, \ldots, k\}$) to keep track of this best packing. Their proposed heuristic then performs as described in Algorithm 2 below.

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 $S(k,r) = \emptyset$ 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 $r + w_k \leq c$ **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 $r^* = \arg\max_{0 \leq r \leq c} f(n,r)$.
Output the set $S(n,r^*)$ and the associated profit $f(n,r^*)$.

This algorithm runs in $\mathcal{O}(n^2c)$ time and uses $\mathcal{O}(n^2c)$ of memory in order to store the set $S(k,r)$. This memory requirement can be reduced to $\mathcal{O}(nc)$, see [13] for details. The algorithm is then strengthened using some sorting of the items at the beginning, some tie-breaking rules (when β is calculated) and a 'fill-and-exchange' local search at the end to yield a great deterministic heuristic algorithm for the QKP.

2.3 Other existing heuristics for the QKP

Prior to the DP heuristic of Djeumou-Fomeni and Letchford [13], several primal heuristic algorithms have been devised for the QKP, but mainly for the purpose of obtaining a rapid feasible solution that could be used in exact solution methods. In these lines, Gallo et al. [15] used the idea of upper planes to find an upper bound to the profit contribution of each item and then solved a linear knapsack problem, wherein the profit of each item is its corresponding upper plane. The solution of such a knapsack problem is clearly a feasible solution for the QKP. They were also the first to propose the idea of the local search procedure called 'fill-up-and-exchange' to the context of the QKP. This procedure consists of either adding one item to the knapsack, or exchanging one item in the knapsack

for one item outside. Chaillou et al. [10] presented a greedy heuristic, which starts by sorting all the items in the non-decreasing order of their 'loss-to-weight' ratio δ_i/w_i, where δ_i represents the decrease in the profit that would be incurred if item i were removed, then placing all the items in the knapsack, and finally removing them iteratively (following the ordering initially established) until feasibility is achieved. Elsewhere, Billionet & Calmels [6] presented a hybrid method, in which the method of Chaillou et al. [10] is used to form an initial solution, and then the fill-up-and-exchange procedure of Gallo et al. [15] is used to improve that solution. Some other more complex heuristics have been proposed, based for example on Lagrangian relaxation [9] and tabu search [16]. For details, we refer the reader to [24].

In the more recent literature of the QKP, approximation algorithms have been proposed in [26, 31, 32]. On the other hand, recent metaheuristic algorithms have been presented for the QKP. Chen and Hao [11] proposes an iterated "hyperplane exploration" approach, which adopts the idea of searching over a set of hyperplanes defined by a cardinality constraint to delimit the search to promising areas of the solution space. This method also combines Tabu Search to locate high quality solutions within the reduced solution space. Patvardhan et al. [27] also proposed a parallel improved quantum inspired evolutionary algorithm for the QKP. We refer interested readers to these two references of other existing meta-heuristics for the QKP. Primal heuristic solutions that are obtained by rounding the fractional solution of a continuous relaxation have been used in [12, 14].

It should be noted that most of these existing heuristic algorithms have reported results solely based on standard QKP instances that are usually generated following the scheme initially proposed by Gallo et al. [15]. However, in a recent research, Schauer [30] showed that such standard instances may sometimes be relatively easy for modern heuristic algorithms. A set of problem instances that can be formulated as QKP is then presented in their work and shown to be very challenging to many existing QKP heuristic algorithms. In this paper, our computational results show that the proposed lifted DP algorithm can perform admirably well on a large proportion of the hard instances presented by Schauer [30].

2.4 Exact solution methods for the QKP

Over the past few decades, several solution algorithms have been developed for the QKP. We refer interested reader to the surveys by Kellerer et al. [20] and by Pisinger [24] for a good review of some of these approaches. One of the most effective exact algorithms for the QKP is that of Caprara et al. [9], which is based on Lagrangian relaxation, subgradient optimization, and branch-and-bound. It can quickly solve instances with up to 400 items when the profit matrix is fully dense, and instances with up to 200 items for low density profit matrix. This algorithm was made even more effective by Pisinger et al. [25], using several powerful reduction techniques. There are also some algorithms by Billionnet and Soutif [7, 8] that are based on integer programming linearization and on Lagrangian relaxation. These algorithms produce the opposite effect of Caprara's algorithm in the sense that they perform well for low density QKP instances and have less interesting results on high density QKP instances. More recently, Djeumou-Fomeni et al. [12] proposed a cut-and-branch algorithm with reported results for instances with up to 800 items regardless of the density of the problem. It is clear that these exact algorithms are all limited by either the size of the problems that can be solved, or the amount of time needed to solve these instances.

3 Lifted DP heuristic for the QKP

If we were to adapt the classical dynamic programming algorithm (see Algorithm 1) to the QKP, we could define $f(k,r)$ in a natural way, as:

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

It has been shown in [13] that, using this definition, there is no analogue of the recursive Equation (4).

The main contribution of this paper is to device a heuristic algorithm based on the idea of the DP, but which will operate in the space of the lifted variables to yield an efficient heuristic solution for the QKP. In order to do this, we consider the complete graph $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{N} = \{1, \ldots, n\}$ is the set of nodes and corresponds to the set of items in the original problem (1)–(3), and $\mathcal{E}$ is the set of edges which represents any pair of chosen items. Notes that $\mathcal{E} = \{e = (i, j) : \forall i, j \in \mathcal{N}, i < j\}$. The objective function (1) can be written as

$$\sum_{i=1}^{n} q_{ii} x_i + 2 \sum_{e \in \mathcal{E}} q_{ij} y_e,$$

where $y_e = x_i x_j$ for $e = (i, j)$. The new variables y_e are the lifted space variables, which will be used in the proposed Lifted DP algorithm. It should be noted that lifted space variables are often used for the linearization of quadratic 0–1 optimization problem, see for example [17, 18, 28, 29, 33]. But, to the best of our knowledge, it has not been used previously in the context of DP algorithm. The use of these variables allows us to capture both the contribution of the items and that of the pairs of items within the DP algorithm.

Since the Bellman's principle of optimality cannot hold for the QKP (see [13]), it will be helpful to re-define $f(k, r)$ 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 $f(k, r)$. Note that these are the same definitions used in [13]. However, in this paper, we will extend the index k to also cover the first k pairs of items (edges). Thus, we will analogously define $f(e, r)$ and $S(e, r)$ when we consider the pairs of items represented by the edges of our complete graph.

The implementation of the DP algorithm with both the linear space variables (the items) and the lifted space variables (the pairs of items) requires to consider the profit and the weight contributions of both at each iteration. As far as an item k is concerned, its profit contributions can be calculated as $q_k + 2 \sum_{i \in S(k-1, r-w_k)} q_{ik}$, with its weight contribution simply being w_k. However, when it comes to the edges, one has to be careful, since one or both the items forming the edge might have already been included into the knapsack. Thus, for a given edge $e = (i, j)$, its weight and profit contributions can be calculated as follows:

$$w(e) = \begin{cases} w_i + w_j & if \quad i, j \notin S(e-1, r - w_i - w_j) \\ w_i & if \quad i \notin S(e-1, r - w_i) \text{ and } j \in S(e-1, r - w_i) \\ w_j & if \quad i \in S(e-1, r - w_j) \text{ and } j \notin S(e-1, r - w_j) \\ 0 & if \quad i, j, \in S(e-1, r) \end{cases}$$

while its profit contribution can be calculated as:

$$q_e = \begin{cases} q_{ii} + q_{jj} + 2 \sum\limits_{i' \in S(e-1, r-w_i-w_j)} (q_{i'i} + q_{i'j}) & if \quad i, j \notin S(e-1, r - w_i - w_j) \\ q_{ii} + 2 \sum\limits_{i' \in S(e-1, r-w_i)} q_{i'i} & if \quad i \notin S(e-1, r - w_i) \text{ and } j \in S(e-1, r - w_i) \\ q_{jj} + 2 \sum\limits_{i' \in S(e-1, r-w_j)} q_{i'j} & if \quad i \in S(e-1, r - w_j) \text{ and } j \notin S(e-1, r - w_j) \\ 0 & if \quad i, j \in S(e-1, r) \end{cases}$$

We can now present our heuristic algorithm which performs as described in Algorithm 3 below.

The contributions of the *for loops* in Line 3 and Line 4 of the algorithm to the complexity of the algorithm are $\mathcal{O}(n)$ and $\mathcal{O}(c)$, respectively. Similarly, the *for loops* in Line 17 and Line 18 contribute for $\mathcal{O}(n^2)$ and $\mathcal{O}(c)$, respectively. Finally, one should note that the quantities β and $\beta(e)$ used in Algorithm 3 can be computed in linear time, using the following identities:

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

and

$$\beta(e) = f(e-1,r) + \begin{cases} q_e + q_{ii} + q_{jj} + 2\sum\limits_{i' \in S(e-1,r)} (q_{i'i} + q_{i'j}) & if \quad i,j \notin S(e-1,r) \\ q_{ii} + 2\sum\limits_{i' \in S(e-1,r)} q_{i'i} & if \quad i \notin S(e-1,r) \text{ and } j \in S(e-1,r) \\ q_{jj} + +2\sum\limits_{i' \in S(e-1,r)} q_{i'j} & if \quad i \in S(e-1,r) \text{ and } j \notin S(e-1,r) \\ 0 & if \quad i,j \in S(e-1,r) \end{cases}$$

Therefore, the overall time complexity of this algorithm is $\mathcal{O}(n^3c)$. Elsewhere, the memory space required to store the set $S(k,r)$, for $k = 1,\ldots,n+|\mathcal{E}|$, and $r = 0,\ldots,c$ is $\mathcal{O}(n^3c)$. This memory storage complexity is a major drawback to Algorithm 3 as it means that only instances with very limited sizes can be solved on an ordinary computer. In the next section, we will show that it is possible to reduce the memory requirement to $\mathcal{O}(nc)$ without worsening the running time of the algorithm.

Algorithm 3 :Lifted dynamic programming for the QKP

Initialise $f(0,0)$ to 0, and $f(k,r)$ to $-\infty$ for all other k,r.
Initialise $S(k,r) = \emptyset$ 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 $r + w_k \leq c$ **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
for $e = 1,\ldots,|\mathcal{E}|$ **do**
  for $r = 0,\ldots,c$ **do**
    if $f(e-1,r) > f(e,r)$ **then**
      Set $f(e,r) := f(e-1,r)$ and $S(e,r) := S(e-1,r)$.
    end if
    if $r + w(e) \leq c$ **then**
      Let $\beta(e)$ be the profit of $S(e-1,r) \cup \{e\}$.
      if $\beta(e) > f(e,r+w(e))$ **then**
        Set $f(e,r+w(e)) := \beta$,
        Set $S(e,r+w(e)) := S(e-1,r) \cup \{e\}$.
      end if
    end if
  end for
end for
Let $(k^*,r^*) = \arg \max\limits_{\substack{0\leq r\leq c,\\ 1\leq k\leq n+|\mathcal{E}|}} f(k,r)$.
Output the set $S(k^*,r^*)$ and the associated profit $f(k^*,r^*)$.

4 Memory reduction and local search

This section is dedicated to some improvements of Algorithm 3 presented above. First, we show an implementation version of the algorithm that requires less memory storage. Then we present a modified version of the 'fill-up-and-exchange' local search procedure that aims at improving the solution obtained from the lifted space DP heuristic.

4.1 Reducing the memory requirement of the algorithm

The reduction of the memory requirement of Algorithm 3 will come from the observation that one can drop the index k from the state function $f(k,r)$ for $k = 1,\ldots,n+|\mathcal{E}|$, and $r = 0,\ldots,c$. Indeed, note that all of the computations carried out in any given stage depend only on the values that were computed in the preceding stage. For this reason, we do not need to store the calculation of all the $f(k,r)$ along with their corresponding sets $S(k,r)$. Instead, we will define $f(r)$ to be the *current* value of the most profitable packing found so far, regardless of the stage k, having a total weight of r. This implies that for each $r = 0,\ldots,c$ and each $i = 1,\ldots,n$, we also define the a Boolean variable $B(r,i)$ to take the value 1 if item i is packed in the set of items that provides a profit $f(r)$. The implication of these new definition to our algorithm is that, one now needs to iteratively decrease the value of r instead of increasing it as in Algorithm 3. In fact, the value of $f(r)$ at any given stage k, for $k = 1,\ldots,n+|\mathcal{E}|$ 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 $r' > r$. The reduced-memory version of our heuristic algorithm then performs as described in Algorithm 4 below.

Algorithm 4 : Reduced memory Dynamic programming

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 $r \geq w_k$ **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
for $e = 1,\ldots,|\mathcal{E}|$ **do**
    for $r = c$ to 0 (going down in steps of 1) **do**
        if $r \geq w_e$ **then**
            Let $\beta(e)$ be the profit of $S(e-1, r-w_e) \cup \{e\}$.
            if $\beta(e) > f(r)$ **then**
                Set $f(r) := \beta(e)$,
                Set $B(r,i) := 1$ and $B(r,j) := 1$,
                where $e = (i,j)$.
            end if
        end if
    end for
end for
Let $r^* = \arg\max_{0\leq r\leq c} f(r)$.
Compute the final set of items $S(k^*, r^*)$.
(This can be done in $\mathcal{O}(n)$ time using the $B(r^*,i)$ values.)
Output $S(k^*, r^*)$ and the associated profit $f(r^*)$.

It should be noted that this idea of reducing the memory requirement by dropping the k index in the calculation $f(k,r)$ for $k = 1,\ldots,n+|\mathcal{E}|$, and $r = 0,\ldots,c$ is an adaptation of the one presented in [13]. Furthermore, the modified algorithm now still runs in $\mathcal{O}(n^3c)$ time, but only requires $\mathcal{O}(nc)$ of memory storage. This makes the algorithm capable of solving large scale instances.

4.2 A modified 'fill-up-and-exchange' local search

The traditional 'fill-up-and-exchange' local search procedure has for long been used in heuristic algorithms for the linear KP. It was first introduced in the context of the QKP by Gallo et al. [15]. This local search procedure is often used to improved the solution to a knapsack-like problem obtained by a heuristic method. Its first part consists of filling up the residual knapsack space with any item that is not included in the packing of the heuristic solution. Then, the second part consists of looking for possible swap between a packed item and an unpacked one, if this allows an improvement of the profit value.

In this paper, we implement a slightly modified version of this procedure by adapting it to the lifted space variables. More precisely, the 'fill-up' phase is extended to the consideration of edges, in the sense that, we also look for the possibility of filling-up the residual capacity of the knapsack with pairs of items. In the 'exchange' phase, we consider the following swapping options:

- a packed item with a unpacked one,
- a packed item with an unpacked edge,
- a packed edge with an unpacked item.

Note, that one of the reasons why we do not consider the option of swapping a packed edge with an unpacked one is that it will take the overall complexity of the algorithm beyond the $\mathcal{O}(n^3c)$ running time. Our computational results show that this modified 'fill-up-and-exchange' procedure has a significant impact on the final results obtained by the overall proposed algorithm.

5 Computational experiments and results

In this section, we present our computational results. We coded all our routines in the C programming language and compiled them with *gcc 4.6*. All the results were obtained by running our code on a single cluster node of the super computer of Compute Canada[1] with processor at 2.26 GHz and 24 GB of RAM. We will first describe the test instances that were used and then present and analyze the obtained results

5.1 Description of the test instances

Most of the existing literature on QKP heuristic algorithms only report results for "standard" QKP instances that are obtained using the scheme first introduced by Gallo et al. [15]. These instances are generated as follows. For a given value of n, each weight w_i is an integer uniformly distributed between 1 and 100. The knapsack capacity c is an integer uniformly distributed between 50 and $\sum_{i\in N} w_i$. Finally, for a given choice of *density parameter* $\Delta\%$, each profit term q_{ij} is set to zero with probability $(100-\Delta)\%$, and set to an integer uniformly distributed between 1 and 100 with probability $\Delta\%$.

To test our algorithm, we created a total of 680 instances, which corresponds to 10 random instances for each combination of $n \in \{50, 100, 150, \ldots, 850\}$ and $\Delta \in \{25\%, 50\%, 75\%, 100\%\}$.

An asymptotic study of the QKP [25, 30] has shown that such standard instances may sometimes be relatively easy for modern heuristic algorithms. Therefore, we also consider several other families of instances, as described in the following subsections.

[1] www.computecanada.ca

5.1.1 Dispersion problem instances

The dispersion problem consists of locating q facilities at n possible locations, while maximizing the sum of the pairwise distances between facilities. The QKP formulation of this problem is as follows [25]:

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

where d_{ij} is the distance between locations i and j. Several variants of this problem, which differ by the distribution of the d_{ij}, are proposed in [25]:

- *GEO* (*Geometrical problems*): the n locations are randomly located in a 100×100 square, and d_{ij} is the Euclidean distance between locations i and j.
- *WGEO* (*Weighted geometrical problems*): the locations are again randomly located in a square, but each location i is assigned a weight α_i in the interval $[5, \ldots, 10]$. The distance d_{ij} is then $\alpha_i \alpha_j$ times the Euclidean distance between locations i and j.
- *EXPO* (*Exponential problems*): for each pair $\{i, j\}$ of locations, d_{ij} is randomly drawn from a negative exponential distribution with mean 50.
- *RAN* (*Random problems*): for each pair $\{i, j\}$, d_{ij} is uniformly distributed in $[1, \ldots, 100]$.

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

Pisinger et al. [25] also presented a knapsack-like version of the four above-mentioned problem types. These are obtained by generating a weight w_i (randomly in $[1, \ldots, 100]$) for each location i, setting q to $\lfloor \frac{1}{2} \sum_{i=1}^{n} w_i \rfloor$, and changing the knapsack constraint accordingly. These instances will be denoted $KP - \{EXPO, GEO, WGEO, RAN\}$. We generated ten instances for each combination of problem type and $n \in \{25, 50, 100, 200, 400\}$, for a total of 400 instances.

5.1.2 Densest subgraph instances

The densest subgraph problem was also formulated as a QKP by Pisinger et al. [25]. Given a graph $G = (V, E)$, this problem amounts to finding a set of nodes $U \subseteq V$ of cardinality q for which the induced subgraph contains the most possible edges. Variants of this problem are obtained by varying the density of G. We experimented with the following settings:

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

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

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

5.1.3 Hidden-clique instances

The final set of test instances used in this paper is the so-called "hidden clique" (HC) instances. These instances were recently introduced in the context of the QKP by Schauer [30], who showed that they are extremely challenging for existing QKP heuristic algorithms. In fact, this class of problems on its own make up a completely different field of research [1, 2]. For a given n, one generates a random

(so-called Erdős-Rényi) graph, in which each edge is present with probability 1/2. One then "hides" a clique in it, by selecting a random set of $\lfloor n \rfloor$ nodes, and adding edges, where necessary, so that those nodes form a clique. The knapsack capacity is then set to $\lfloor n \rfloor$. The weight of each vertex is 1, its linear profit is 0 and the quadratic profit is 1 whenever an edge is present in the graph, and 0 otherwise. The optimal solution value is then almost surely $\frac{1}{2}\lfloor n \rfloor (\lfloor n \rfloor - 1)$. We generated 10 HC instances for each value of $n \in \{20, 40, 60, \ldots, 400\}$, making 200 in total.

5.2 Results for the standard instances

Our first set of experiment is carried out on standard QKP instances. The results for this experiment are reported in Table 1. In the experiment, we were interested in three particular aspects. Firstly, we wanted to know how good is our Lifted DP algorithm alone ('Lifted DP'), i.e., without the local search component. Secondly, we wanted to know the contribution of our local search procedure to the overall performance of the algorithm ('Lifted DP + FE'). Finally, we wanted to see how our new Lifted DP algorithm compares with the DP heuristic algorithm of Djeumou-Fomeni and Letchford [13] ('FL2014'), which is well known in the literature of the QKP as one of the state-of-the-art deterministic heuristic algorithms.

The first two columns of Table 1 show, respectively, the density of the profit matrix and the size of the problem defined in term of the number of items. The following three columns show the average, out of of 10 instances, of the percentage gap of the 'Lifted DP', the number of instances (out of 10) for which the algorithm found an optimal solution and the average CPU time taken by the algorithm, respectively. It should be noted that the optimality gap is calculated as:

$$\left(\frac{\text{Optimal value} - \text{Heuristic value}}{\text{Optimal value}} \right) \times 100.$$

In order to obtain an optimal solution for each of these instances, we used the cut-and-branch algorithm of Djeumou-Fomeni et al. [12]. The next three columns report the same metrics as the latter three, but for our overall proposed algorithm 'Lifted DP + FE'. In the last column of this table, we report the average percentage gap obtained with the DP heuristic algorithm of [13] for the same instances. We should point out that the running time of the latter algorithm is not reported because, theoretically it is much faster than the proposed algorithm by a factor of n, ($\mathcal{O}(n^3c)$ vs $\mathcal{O}(n^2c)$).

The results in Table 1 show that when the Lifted DP algorithm, described in Algorithm 1 is used alone, it is able to find an optimal solution to about 90% of the 680 problem instances. This percentage goes up to more than 97% of the 680 instances when the Lifted DP algorithm is combined with our modified 'fill-up-and-exchange' local search procedure. It can also be noticed that for the instances for which our algorithm could not find an optimal solution, the relative percentage gap is consistently below 0.01%. A comparison with the algorithm of Djeumou-Fomeni and Letchford [13] ('FL2014') shows that our proposed algorithm is dominant in terms of optimality gap, especially for the low density instances. While the algorithm Djeumou-Fomeni and Letchford [13] shows some less-interesting results for low density instances. When it comes to the CPU time, it can be seen that our proposed algorithm is a bit time consuming, as it can take up to one hour to solve some of the large instances. This is conceptually expected given that our algorithm runs in $\mathcal{O}(n^3c)$ times compares to the $\mathcal{O}(n^2c)$ times for the DP heuristic of [13].

5.3 Results for the dispersion and densest subgraph instances

In the next set of our experiments, we apply our proposed algorithm to the 600 instances of the dispersion and densest subgraph problems. The results for these experiments are reported in Table 2, in which the metrics reported are the same as the ones in Table 1. We should also point out here that the optimal solutions used to calculate the optimality gaps were also obtained using the cut-and-branch algorithm of [12].

Table 1: Results for the standard QKP instances

Instance		Lifted DP			Lifted DP + FE			FL2014
Density	size n	Gap (%)	# Opt./10	Time (s)	Gap (%)	# Opt./10	Time (s)	Gap (%)
	50	0.084	9	0.107	**0.000**	10	0.108	0.110
	100	**0.000**	10	1.590	**0.000**	10	1.603	0.080
	150	**0.000**	10	7.816	**0.000**	10	7.876	0.070
	200	**0.005**	6	29.674	**0.005**	6	29.842	0.020
	250	**0.000**	10	65.568	**0.000**	10	65.963	0.010
	300	**0.005**	9	125.104	**0.005**	9	126.190	0.060
	350	0.002	8	208.101	**0.000**	10	210.442	0.050
	400	**0.000**	10	527.510	**0.000**	10	531.026	0.010
$\Delta = 25\%$	450	0.000	10	679.941	0.000	10	684.713	0.000
	500	0.000	9	908.312	0.000	10	916.343	0.000
	550	0.008	9	1,408.717	0.000	10	1,421.072	0.000
	600	0.000	9	2,331.683	0.000	9	2,349.155	0.000
	650	0.001	9	3,383.180	0.000	10	3,414.175	0.000
	700	0.000	10	1,404.734	0.000	10	1,436.883	0.000
	750	**0.000**	10	2,936.410	**0.000**	10	2,985.113	0.010
	800	0.001	8	1,918.941	**0.000**	10	1,993.617	0.050
	850	**0.000**	8	2,811.013	**0.000**	10	3,003.241	0.030
	50	**0.000**	10	0.205	**0.000**	10	0.206	0.290
	100	**0.002**	9	2.447	**0.002**	9	2.475	0.020
	150	**0.009**	7	13.410	**0.009**	7	13.529	0.010
	200	0.031	9	26.368	**0.000**	10	26.714	0.010
	250	**0.000**	9	118.965	**0.000**	9	119.872	0.020
	300	0.000	10	258.679	0.000	10	260.463	0.000
	350	0.000	10	513.177	0.000	10	516.820	0.000
	400	0.000	10	1,190.313	0.000	10	1,195.872	0.000
$\Delta = 50\%$	450	0.000	9	1,301.650	0.000	9	1,310.107	0.000
	500	0.000	10	1,239.945	0.000	10	1,248.338	0.000
	550	**0.000**	10	1,476.207	**0.000**	10	1,489.364	0.020
	600	0.000	9	2,401.845	0.000	10	2,419.569	0.000
	650	0.000	10	3,290.070	0.000	10	3,313.781	0.000
	700	0.000	9	3,499.408	0.000	10	3,541.074	0.000
	750	0.000	10	3,282.415	0.000	10	3,338.002	0.000
	800	0.000	10	3,438.049	0.000	10	3,507.950	0.000
	850	0.001	8	3,866.585	**0.000**	10	3,957.478	0.010
	50	0.000	10	0.139	0.000	10	0.140	0.000
	100	0.004	9	2.157	0.004	9	2.173	**0.000**
	150	0.001	9	8.220	**0.000**	10	8.310	0.010
	200	**0.001**	9	31.344	**0.001**	9	31.583	0.010
	250	0.002	9	54.055	0.000	10	54.737	0.000
	300	0.000	10	173.169	0.000	10	174.484	0.000
	350	0.000	10	206.817	0.000	10	209.471	0.000
	400	0.000	10	585.993	0.000	10	590.070	0.000
$\Delta = 75\%$	450	0.000	10	1,040.464	0.000	10	1,047.979	0.000
	500	0.000	10	1,848.305	0.000	10	1,857.684	0.000
	550	0.000	8	1,564.884	0.000	9	1,583.896	0.000
	600	0.000	10	3,547.376	0.000	10	3,567.487	0.000
	650	0.000	10	2,386.271	0.000	10	2,411.889	0.000
	700	0.000	10	2,364.701	0.000	10	2,400.335	0.000
	750	**0.000**	10	3,628.308	**0.000**	10	3,682.535	0.010
	800	0.000	10	3,298.709	0.000	10	3,367.395	0.000
	850	0.000	8	2,911.023	0.000	10	3,013.531	0.000
	50	0.009	9	0.092	0.009	9	0.093	**0.000**
	100	**0.000**	10	1.430	**0.000**	10	1.440	0.010
	150	0.000	10	7.841	0.000	10	7.892	0.000
	200	0.000	9	27.588	0.000	9	27.821	0.000
	250	0.000	10	43.790	0.000	10	44.232	0.000
	300	0.001	8	149.715	0.001	8	150.494	**0.000**
	350	0.001	8	274.513	0.000	8	276.266	0.000
	400	**0.000**	9	258.210	**0.000**	9	260.903	0.020
$\Delta = 100\%$	450	0.000	10	848.278	0.000	10	854.149	0.000
	500	0.000	10	1,328.395	0.000	10	1,336.192	0.000
	550	0.000	9	1,382.499	0.000	9	1,393.449	0.000
	600	0.000	10	1,790.055	0.000	10	1,806.159	0.000
	650	0.000	10	2,188.107	0.000	10	2,213.680	0.000
	700	0.000	10	1,890.779	0.000	10	1,920.263	0.000
	750	0.000	10	1,952.128	0.000	10	2,091.559	0.000
	800	0.000	10	1,290.924	0.000	10	1,331.200	0.000
	850	0.000	10	1,900.788	0.000	10	1,954.333	0.000

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

Instance		Lifted DP			Lifted DP + FE			FL2014
Type	n	Gap (%)	# Opt./10	Time (s)	Gap (%)	# Opt./10	Time (s)	Gap (%)
DSUB25	25	2.173	7	0.004	**0.119**	9	0.004	1.07
	50	0.328	9	0.092	**0.000**	10	0.100	0.09
	100	2.498	5	1.018	**0.074**	8	1.170	9.96
	200	0.702	3	9.633	**0.282**	7	11.836	8.54
	400	0.341	1	294.201	0.005	7	366.096	**0.00**
DSUB50	25	1.014	8	0.005	**0.100**	9	0.006	0.56
	50	0.912	6	0.049	0.118	8	**0.057**	0.86
	100	1.740	4	0.555	**0.098**	7	0.674	7.94
	200	0.209	5	10.239	**0.001**	9	12.532	9.70
	400	0.268	2	289.670	**0.026**	8	354.344	0.19
DSUB75	25	0.636	8	0.003	**0.000**	10	0.004	1.09
	50	0.342	7	0.043	**0.032**	8	0.050	0.34
	100	0.520	5	0.508	**0.017**	7	0.610	3.67
	200	0.307	2	10.548	**0.002**	8	13.256	4.93
	400	0.196	2	225.213	**0.001**	9	285.040	1.59
DSUB90	25	0.122	9	0.006	**0.000**	10	0.006	0.00
	50	0.117	8	0.084	**0.034**	9	0.091	0.15
	100	0.336	5	0.794	**0.054**	7	0.930	2.70
	200	0.071	7	11.695	**0.010**	8	13.227	1.89
	400	0.053	5	258.968	0.000	10	312.130	0.00
EXPO	25	0.222	9	0.004	**0.000**	10	0.005	0.17
	50	0.374	5	0.060	**0.144**	8	0.070	0.43
	100	0.504	2	0.865	**0.223**	5	0.993	3.76
	200	0.590	3	17.125	**0.012**	8	19.613	8.45
	400	0.095	1	336.780	0.000	8	432.479	0.00
GEO	25	0.012	9	0.003	**0.000**	10	0.003	0.14
	50	0.000	10	0.042	0.000	10	0.047	0.00
	100	0.001	9	0.838	**0.000**	10	0.930	0.42
	200	**0.000**	10	14.834	**0.000**	10	16.193	2.22
	400	**0.000**	10	259.289	**0.000**	10	279.633	5.48
WGEO	25	0.000	10	0.020	0.000	10	0.020	0.00
	50	**0.000**	10	0.281	**0.000**	10	0.288	0.02
	100	0.000	10	4.357	0.000	10	4.458	0.00
	200	0.000	10	70.711	0.000	10	72.352	0.00
	400	0.000	10	1,310.435	0.000	10	1,339.523	0.00
RAN	25	0.343	8	0.005	**0.000**	10	0.006	0.50
	50	0.388	3	0.087	**0.131**	6	0.099	0.42
	100	0.325	3	0.804	**0.136**	7	1.039	8.21
	200	0.181	4	23.466	**0.005**	9	27.335	4.88
	400	0.336	1	286.266	**0.050**	8	357.078	13.92
KP-EXPO	25	0.000	10	0.129	0.000	10	0.129	0.00
	50	0.000	10	2.498	0.000	10	2.504	0.00
	100	0.000	10	39.272	0.000	10	39.373	0.00
	200	0.000	10	139.071	0.000	10	139.396	0.00
	400	0.000	10	1,474.347	0.000	10	1,477.855	0.00
KP-GEO	25	0.000	10	0.141	0.000	10	0.141	0.00
	50	0.000	10	2.416	0.000	10	2.423	0.00
	100	0.000	10	39.276	0.000	10	39.380	0.00
	200	0.000	10	135.379	0.000	10	135.706	0.00
	400	0.000	10	1,525.159	0.000	10	1,528.664	0.00
KP-WGEO	25	0.000	10	0.154	0.000	10	0.154	0.00
	50	0.000	10	2.615	0.000	10	2.623	0.00
	100	0.000	10	45.139	0.000	10	45.257	0.00
	200	0.000	10	153.943	0.000	10	154.311	0.00
	400	0.000	10	1,644.290	0.000	10	1,648.221	0.00
KP-RAN	25	0.000	10	0.139	0.000	10	0.140	0.00
	50	0.000	10	2.470	0.000	10	2.477	0.00
	100	0.000	10	40.377	0.000	10	40.480	0.00
	200	0.000	10	174.943	0.000	10	174.612	0.00
	400	0.000	10	1,848.668	0.000	10	1,876.927	0.00

The results in Column 'FL2014' of Table 2 reveal that the DP heuristic algorithm of Djeumou-Fomeni and Letchford [13] performs very badly for most of these instances, especially for the "plain" dispersion and densest subgraph instances, where the optimality gap can go up to 13%. This can be seen as a confirmation of the challenging property of these instances as pointed out in [25, 30]. However, when it comes to our proposed Lifted DP heuristic, combined with the modified local search procedure, the results in Column 'Lifted DP + FE' show that it can find an optimal solution to nearly 92% of all the 600 instances tested. Moreover, for the instances for which optimal solutions could not be found, our algorithm produces feasible solutions that are within 0.1% of optimality. It can also be noted that the computational times needed by our algorithm to solve these instances are relatively low, except for some of the Knapsack-like instances. Indeed, given that our algorithm runs in $\mathcal{O}(n^3c)$ times, the values of c for these instances are relatively large compared to their values for the other 'plain" dispersion and densest subgraph instances, where $c < n$. Interestingly, for these Knapsack-like instances, both our proposed algorithm and the DP heuristic of [13] could find optimal solutions for all the instances tested. This can be explained by the fact that the knapsack nature of these instances make them much closer to the standard QKP instances.

5.4 Results for the Hidden clique instances

Finally, we consider the 200 Hidden clique (HC) instances. For these instances the optimal solution are known in advance when they are generated. Therefore an exact solution algorithm is not required in this set of experiments. We first solved the instances using our proposed Lifted DP heuristic with the modified 'fill-up-and-exchange' procedure ('Lifted DP + FE') and then re-solve the same instances using the DP heuristic of [13] (FL 2014). The results are reported in Figure 1, in which we have plotted the optimal solution values for all the instances, as well as the solution values for both of the above-mentioned heuristics. On the $x-$axis, we have the sizes of the instances n, while the $y-$axis gives the objective function values.

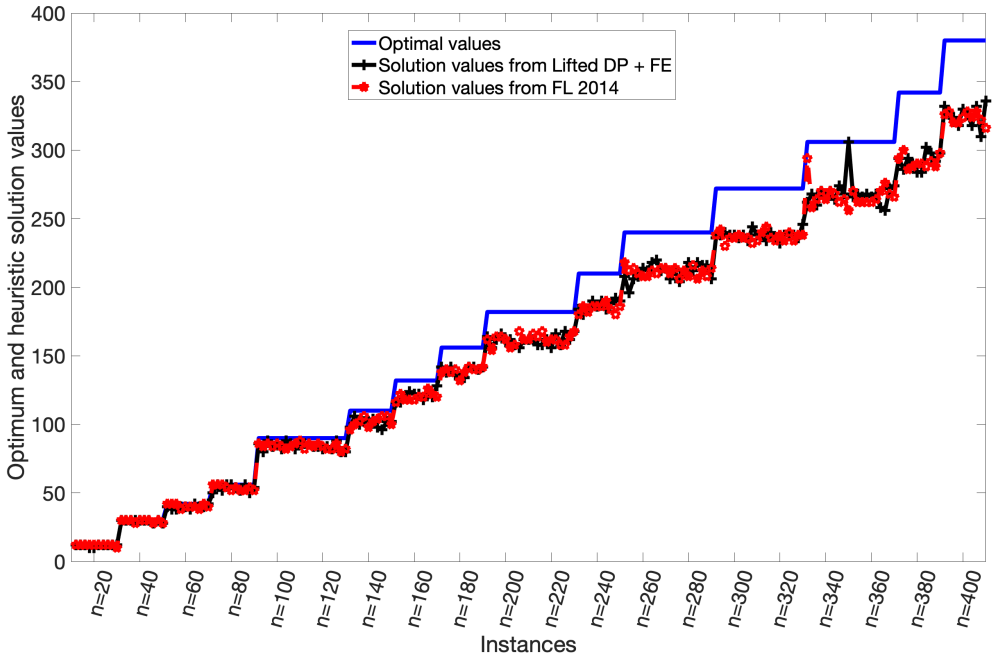

Figure 1: Results for planted clique instances

It can be seen in Figure 1 that for instances with up to 160 items, the heuristic solution values from both algorithms are pretty close to the optimal solution values. While, for instances with 180 items and more there is a significant gap between the optimal solution value and heuristic ones. These

gaps can go up to about 10%. In comparison with one another, there does not appear to be any significant difference in performance for both heuristic algorithms. Their poor performances on these test instances further strengthen the fact fact that Hidden Clique instances seen as QKP are really hard instances for QKP algorithm. However, the fact that this class of problems on its own make up a completely different field of research [1, 2] should not be ignored. Particularly, because most QKP algorithms are designed to solve QKPs.

6 Conclusion

The QKP is an important combinatorial optimization problems with many applications, but for which existing solution algorithms are very limited in their capabilities. Our contribution in this paper has been to propose a novel heuristic algorithm for this strongly $\mathcal{NP}-$hard problem. Our algorithm consists of an adaptation of dynamic programming approach combined with an adaptation of the 'fill-up-and-exchange' local search procedure, with the particularity that both are implemented in the space of the lifted QKP variables. This algorithm runs in $\mathcal{O}(n^3c)$ times and uses $\mathcal{O}(nc)$ memory space. The advantage of moving up in the space of lifted variables is that one can more efficiently capture some of the quadratic attributes of the problem and obtain much better quality solutions than with existing heuristic solutions. We have conducted a thorough computational experiment with a total of 1480 QKP instances, including both the standard and known-challenging QKP instances. The results show that our algorithm can find an optimal solution to more than 97% of the standard QKP instances. It can also find an optimal solution to more than 80% of the challenging QKP instances. For most of the instances for which optimality could not be found, our algorithm has been able to produce feasible solutions within 0.1% (on average) of optimality. It has been clear in our experiments that our adapted version of the 'fill-up-and-exchange' local search procedure contributed significantly to the quality of our solutions.

Elsewhere, we have also presented some comparisons between our proposed algorithm and one of the known state-of-the-art deterministic heuristic algorithm for the QKP [13]. The results have shown that our proposed algorithm is largely superior in terms of the quality of the obtained solutions. This superiority is even more significant on some of the challenging QKP instances, namely, the dispersion and densest subgraph instances. However, it remains that the latter algorithm is faster than ours by a factor of n. Furthermore, the class of Hidden Clique problems, taken as QKP still proves to be challenging to our proposed algorithm.

In sum, our newly proposed Lifted DP heuristic algorithm with the modified 'fill-up-and-exchange' local search procedure appears to be an important milestone in the direction of devising efficient solution for the QKP. Thus naturally positioning itself as a good candidate heuristic to be incorporated in exact solution methods. As direction for future research, we believe that it would be interesting to see how this algorithm could impact in an exact solution framework for the QKP, such as a Branch-and-Cut framework.

References

[1] N. Alon, M. Krivelevich & B. Sudakov (1998), Finding a large hidden clique in a random graph. Random Structures and Algorithms, 13, 457–466.

[2] N. Alon, S. Arora, R. Manokaran, D. Moshkovitz & O. Weinstein (2011), Inapproximabilty of densest k-subgraph from average case hardness. Technical report, Computer Science Department, Princeton University, Princeton, NJ, (2011)

[3] E. Balas & E. Zemel (1980) An algorithm for large zero-one knapsack problems. Operations Research, 28, 1130–1154.

[4] R.E. Bellman (1957) Dynamic Programming. Princeton, NJ: Princeton University Press.

[5] A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige & A. Vijayaraghavan (2010) Detecting high log-densities: an $\mathcal{O}(n^{1/4})$ approximation for densest k-subgraph. In L.J. Schulman (ed.) Proceedings of the 42nd ACM Symposium on Theory of Computing, pp. 201–210. ACM Press.

[6] A. Billionet & F. Calmels (1996) Linear programming for the quadratic knapsack problem. European Journal of Operational Research, 92, 310–325.

[7] A. Billionet & E. Soutif (2004) Using a Mixed Integer Programming Tool for Solving the 0–1 Quadratic Knapsack Problem. INFORMS Journal on Computing, 16, 188–197.

[8] A. Billionet & E. Soutif (2004) An exact method based on Lagrangian decompositionfor the 0–1 quadratic knapsack problem. European Journal of Operational Research, 157, 565–575.

[9] A. Caprara, D. Pisinger & P. Toth (1998) Exact solution of the quadratic knapsack problem. INFORMS Journal on Computing, 11, 125–137.

[10] P. Chaillou, P. Hansen & Y. Mahieu (1983) Best network flow bound for the quadratic knapsack problem. Presented at the International Workshop on Network Flow Optimization (NETFLOW), Pisa, Italy, March 1983.

[11] Yuning Chen & Jin-Kao Hao (2017), An iterated "hyperplane exploration" approach for the quadratic knapsack problem. Computers & Operations Research, 77, 226–239.

[12] F. Djeumou Fomeni, K. Kaparis & A. N. Letchford (2020), A cut-and-branch algorithm for the quadratic knapsack problem. Discrete Optimzation, published online (March 2020).

[13] F. Djeumou Fomeni & A. N. Letchford (2014), A dynamic programming heuristic for the quadratic knapsack problem. INFORMS Journal on Computing, 26(1), 173–183.

[14] M. Fampa, D. Lubke, F. Wang & H. Wolkowicz (2020), Parametric convex quadratic relaxation of the quadratic knapsack. European Journal of Operational Research, 281, 36–49.

[15] G. Gallo, P.L. Hammer & B. Simeone (1980) Quadratic knapsack problems. Mathematical Programming Studies, 12, 132–149.

[16] F. Glover & G. Kochenberger (2002) Solving quadratic knapsack problems by reformulation and tabu search, single constraint case. In: P.M. Pardalos, A. Migdalas & R.E. Burkard (Eds.) Combinatorial and Global Optimization, vol. 14. Singapore: World Scientific.

[17] F. Glover & E. Woolsey (1974) Converting the 0–1 polynomial programming problem to a 0–1 linear program. Operations Research, 22, 180–182.

[18] C. Helmberg, F. Rendl & R. Weismantel (2000) A semidefinite programming approach to the quadratic knapsack problem. Journal of Combinatorial Optimization, 4, 197–215.

[19] R.M. Karp (1972) Reducibility among combinatorial problems. In R.E. Miller & J.W. Thatcher (eds.) Complexity of Computer Computations, pp. 85–103. New York: Plenum.

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

[21] H. Kellerer & V.A. Strusevich (2010) Fully polynomial approximation schemes for a symmetric quadratic knapsack problem and its scheduling applications. Algorithmica, 57, 769–795.

[22] S. Khot (2006) Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. SIAM Journal of Computing, 36, 1025–1071.

[23] S. Martello & P. Toth (1990) Knapsack Problems: Algorithms and Computer Implementations. Chichester: Wiley.

[24] D. Pisinger (2007) The quadratic knapsack problem — a survey. Discrete and Applied Mathematics, 155, 623–648.

[25] D. Pisinger, A.B. Rasmussen & R. Sandvik (2007) Solution of large quadratic knapsack problems through aggressive reduction. INFORMS Journal on Computing, 19, 280–290.

[26] Ulrich Pferschy & Joachim Schauer (2016) Approximation of the Quadratic Knapsack Problem. INFORMS Journal on Computing, 28, 308–318.

[27] C. Patvardhan, Sulabh Bansal & A. Srivastav (2016), Parallel improved quantum inspired evolutionary algorithm to solve large size Quadratic Knapsack Problems. Swarm and Evolutionary Computation, 26, 175–190.

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

[29] D.J. Rader Jr. & G.J. Woeginger (2002) The quadratic 0–1 knapsack problem with series-parallel support. Operations Research Letters, 30, 159–166.

[30] J. Schauer (2016) On the rectangular knapsack problem: approximation of a specific quadratic knapsack problem. European Journal of Operational Research, 255, 357–363.

[31] Britta Schulze, Michael Stiglmayr, Luís Paquete, Carlos M. Fonseca, David Willems & Stefan Ruzika (2020) Asymptotic behavior of the quadratic knapsack problem. Mathematical Methods of Operations Research, 92, 107–132.

[32] Richard Taylor (2016) Approximation of the Quadratic Knapsack Problem. Operations Research Letters, 44, 495–497.

[33] Z. Gu, G.L. Nemhauser & M.W.P. Savelsbergh (2000) Sequence-independent lifting in mixed integer programming. Journal of Combinatorial Optimization, 4, 109–129.