Les Cahiers du GERAD ISSN: 0711–2440

Tight upper and lower bounds for the quadratic knapsack problem through binary decision diagram

M. E. Fennich, L. C. Coelho, F. Djeumou Fomeni

G–2024–56

Septembre 2024

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

Tight upper and lower bounds for the quadratic knapsack problem through binary decision diagram

M. Eliass Fennich [a, b]

Leandro C. Coelho [a, c]

Franklin Djeumou Fomeni [a, d]

[a] *GERAD & CIRRELT Montréal (Qc), Canada, H3T 1J4*

[b] *Department of Operations and Decision Systems, Université Laval, Québec (Qc), Canada, G1V AO6*

[c] *Canada Research Chair in Integrated Logistics, Université Laval, Québec (Qc), Canada, G1V AO6*

[d] *Department of Analytics, Operations and Information Technologies, Université du Québec à Montréal, Montréal (Qc), Canada, H2X 3X2*

mohamed-eliass.fennich.1@ulaval.ca
leandro.coelho@fsa.ulaval.caa
djeumou_fomeni.franklin@uqam.ca

Septembre 2024
Les Cahiers du GERAD
G–2024–56

Abstract : The Quadratic Knapsack Problem (QKP) is a challenging combinatorial optimization problem that has attracted significant attention due to its complexity and practical applications. In recent years, Binary Decision Diagrams (BDDs) have emerged as a powerful tool in combinatorial optimization, providing efficient bounds. In the literature of the QKP, all the exact methods are based on computing tight bounds before applying branch-and-bound (B&B) schemes. We advance this literature in this work by leveraging BDDs to compute bounds more effectively. We propose a novel integration of dual bound tightening within a BDD-based B&B framework, employing a Breadth-First Search (BFS) strategy. Our approach addresses the critical limitation of existing BDD-based B&B methods, which often lack robust dual-bound tightening mechanisms. Furthermore, we propose several efficient compilation techniques of BDDs for the QKP. Through an extensive experimentation on several categories of QKP instances, we demonstrate that our method not only competes with but often surpasses the bounding stages of the leading exact algorithms. Notably, our approach reduces the average duality gap by up to 10% for the class of Hidden Clique QKP instances, showcasing its potential. Furthermore, our findings indicate that the BFS B&B method outperforms state-of-the-art BDD B&B approaches across all tested QKP instances, highlighting its effectiveness and potential for broader application.

Keywords : Combinatorial optimization, binary decision diagrams, dynamic programming, branch-and-bound, Quadratic Knapsack Problem

Acknowledgements: This work was partly supported by the Canadian Natural Sciences and Engineering Research Council under grants 2021–03307 and 2019–00094. This support is greatly appreciated. We also thank the Digital Research Alliance of Canada for providing high-performance computing facilities.

1 Introduction

The Quadratic Knapsack Problem (QKP) was introduced in the 1980s and is used to model several applications in fields such as telecommunications, hydrology, and combinatorics (Gallo et al., 1980). Furthermore, in urban planning and logistics, the QKP is used to solve facility location problems (Pisinger et al., 2007), such as those for hospitals and logistics centers. Additionally, the QKP holds a great importance as a theoretical problem in nonlinear optimization, particularly in the context of the clique problem in graph theory, where it is used to identify a strongly connected subgraph of a given size (Caprara et al., 1999; Schauer, 2016).

The QKP is defined with a set $N = \{1, 2, \ldots, n\}$ of items that can be added to a knapsack of integral capacity c. Each item $i \in N$ is characterized by an integer weight ω_i. A symmetric square matrix $P = (p_{ij})_{1 \leq i \leq n, 1 \leq j \leq n}$ of size n represents the profits, where p_{ii} indicates the individual value if item i is selected, and the additional profit $p_{ij} + p_{ji} = 2p_{ij}$ if both items i and j are selected simultaneously, with p_{ij} being non-negative and integral. A formal expression of this problem involves utilizing a variable x_i, defined within the set of binary decisions for each item $i \in N$. The variable x_i indicates whether item i is included in the solution or not.

$$\begin{aligned} \max \quad & \sum_{i \in N} \sum_{j \in N} p_{ij} x_i x_j \\ \text{s.t.} \quad & \sum_{i \in N} \omega_i x_i \leq c, \\ & x_i \in \{0, 1\}, \quad \forall i \in N. \end{aligned} \tag{1}$$

Since its introduction, the QKP has attracted the interest of many researchers, leading to the development of several algorithms. Gallo et al. (1980) were the first to introduce a branch-and-bound (B&B) algorithm based on a family of upper planes, solving instances with a limited number of items. Chaillou et al. (1989) introduced a Lagrangian formulation for the capacity constraint. Billionnet and Calmels (1996) presented a bound based on the linearization of the integrality constraint. These early algorithms only reported results for instances with up to 20 items. This size limit was surpassed by the work of Caprara et al. (1999) and Billionnet et al. (1999). Indeed, Caprara et al. (1999) proposed a Lagrangian relaxation and formulation, reporting optimal results for instances with up to 400 items for dense matrices and 120 for sparse ones. Billionnet et al. (1999) developed a Lagrangian decomposition bounding approach, solving instances of up to 150 items for dense matrices and 300 for sparse ones. However, according to Pisinger et al. (2007), the bounds proposed by Caprara et al. (1999) dominate those of Billionnet et al. (1999), especially for dense instances.

Furthermore, an asymptotic study of the QKP by Schauer (2016) later revealed that randomly generated instances, according to the procedure proposed by Gallo et al. (1980), may be relatively easy to solve. They proposed a new set of instances where most state-of-the-art heuristics failed to report good results. For the exact algorithms, the only reported results for these new instances are by Fomeni et al. (2022), who proposed a bounding of the linearized formulation based on cutting planes. Their approach allowed them to solve instances with up to 800 items for the standard ones (Gallo et al., 1980) and 400 for the recent instances (Schauer, 2016). All the exact algorithms above are of a B&B type, where they compute bounds using different procedures and then use these bounds within an Integer Programming (IP) B&B framework (Létocart et al., 2012). In this paper, we propose a new bounding approach for the QKP based on Binary Decision Diagrams (BDDs), a flexible data structure that allows encoding the solution space of any problem with a dynamic programming (DP) formulation (Bergman et al., 2016).

A BDD is a directed acyclic graph $B = (U, A)$ that encodes the set of values of binary variables x. The set of nodes U represents different states that encode subproblems from the original problem, and the set of arcs A represents transitions between these states. These transitions determine whether the variable x_i is assigned a value of 1 or 0. At each node $u \in U$, there are at most two outgoing arcs: a 0-arc $a_0(u)$ and a 1-arc $a_1(u)$. The nodes are partitioned into layers $L = (l_1, ..., l_{n+1})$, where each

layer l_i corresponds to a set of decisions over the variable x_i for $i \in N$, plus a terminal layer. All arcs in the graph are directed from one layer l_i to another layer l_k with $i < k$. The first and last layers contain only one node each, representing the *root* node (where no variable has been considered) and the *terminal* node (where all the variables have been considered), respectively. A path from the *root* node to the *terminal* node represents a solution s from the set $\text{Sol}(B)$ of the solutions encoded in the graph B. For any two nodes $u, u' \in U$, there is a sub-graph that encodes a set of partial solutions through the paths between u and u'. At each layer, nodes are added and processed to retain only those that encode unique feasible partial solutions. This type of graph is called an exact BDD, and its computation can be very expensive. Approximation approaches are possible to provide bounds on the solution. Bergman et al. (2014a) studied the compilation of an approximated BDD that computes bounds on the optimal solution value instead of directly computing the optimal value by limiting the number of nodes in each layer to a fixed number.

According to Castro et al. (2022), BDDs have existed for over half a century in the Boolean function community. Their application to optimization problems is relatively recent, following some successes in scheduling, routing, and other areas over the past two decades (Castro et al., 2022). Castro et al. (2022) illustrates how BDDs can enhance several existing optimization techniques, such as cutting planes and column generation. Becker et al. (2005) and Tjandraatmadja and Van Hoeve (2019) show that useful cuts can be generated from BDDs and relaxed BDDs if the relaxation is tight. This makes it an attractive approach for enhancing the solution process of complex problems. While BDDs are enumerative algorithms that will list alternative and non-dominated solutions, it is important to define the notion of *restricted* and *relaxed* BDDs for computing bounds:

Restricted BDDs: compute a *primal bound* by restricting the number of nodes in each layer to be below a certain threshold, ignoring some nodes beyond this threshold. This ensures it contains only feasible solutions, although it may miss some or all optimal ones.

Relaxed BDDs: compute a *dual bound* by merging nodes from a layer, such that the relaxed BDD encodes a superset of the set of feasible solutions. This may result in infeasible solutions from the merged nodes being included.

The bounds provided by these BDDs can be made tighter by exploring the trees using a B&B fashion. Bergman et al. (2016) proposed a BDD-based B&B that uses lower and upper bounds computed from restricted and relaxed BDDs iteratively. This is achieved by repeatedly computing restricted BDDs using exact information from relaxed BDDs, i.e., exploring a subset of nodes from the relaxed BDD, called *cutsets*. These cutsets are obtained prior to the merging step of the relaxed BDDs. By exploring these nodes, one can either find new cutsets or improve the primal bound. The relaxed BDDs provide a dual bound for each of these nodes, which can also be made tighter by exploring more nodes from the cutsets. The algorithm of Bergman et al. (2016) iteratively adds nodes (cutsets) to a pool to be explored with restricted BDDs and terminates if all the nodes yet to be explored can be pruned using the primal bound. They reported results showing that this branching approach outperforms commercial solvers on several classes of problems, such as the Maximum Independent Set Problem and the Maximum Cut Problem. As we will see in Section 2.2, their BDD B&B can be seen as a depth-first search (DFS) one. Furthermore, Gillard et al. (2021) proposed a node filtering for BDDs B&B, which reduces the size of the cutsets and accelerates the searching process.

In this paper, we propose a breadth-first search (BFS) B&B method for BDDs, demonstrating that it outperforms the DFS B&B approach, with the added advantage of tightening the dual bound if the algorithm does not finish within a certain time limit, which is not the case in the DFS version. Additionally, we introduce minimum spanning trees (MSTs) as a tool to compute which nodes to merge in relaxed BDDs efficiently. On the QKP side, we propose a DP formulation for BDDs, leveraging fundamental properties and previous results from the QKP literature to compile restricted BDDs efficiently. We compare our bounds from the BFS B&B with the results of the bounding phase of the state-of-the-art exact algorithms previously discussed (Caprara et al., 1999; Fomeni et al., 2022). The experiments show that our approach outperforms these algorithms and computes the best bounds for

all the recent QKP instances (Schauer, 2016). More precisely, we compute near-optimal solutions and tight dual bounds, which reduce 10% the duality gap, leading to a 93% solution quality improvement.

We organize the rest of this paper as follows. Section 2 reviews the state-of-the-art bounds for the QKP of Caprara et al. (1999) and Fomeni et al. (2022) and presents the B&B approach to BDDs of Bergman et al. (2016). In Section 3, we formally introduce the compilation approach of BDDs for the QKP. Sections 4 and 5 describe our approximation approaches for restricted and relaxed BDDs for the QKP, respectively. Section 6 presents the details of our proposed BFS B&B and discusses several implications of this approach. In Section 7, we report the results of the computational experiments used to assess the efficiency of our algorithms. Finally, Section 8 offers some concluding remarks.

2 Literature review

In this section, we describe the bounds from the literature for the QKP and introduce the B&B scheme for exploring BDDs.

2.1 Bounds for the QKP

In this section, we review the bounds computation for the QKP from Caprara et al. (1999) and Fomeni et al. (2022). For a more detailed literature review, we recommend Cacchiani et al. (2022) for the QKP and Castro et al. (2022) for BDDs.

2.1.1 The Caprara, Pisinger, and Toth bounds

The upper bound proposed by Caprara et al. (1999) is computed using the following reformulation of the objective function:

$$\sum_{i\in N}\sum_{j\in N} p_{ij}x_ix_j = \sum_{i\in N}\tilde{p}_i x_i, \tag{2}$$

where $\tilde{p}_i = p_{ii} + \sum_{j\in N\setminus\{i\}} p_{ij}x_j$ and it is bounded by π_i (i.e., $\tilde{p}_i \leq \pi_i$), such that

$$\pi_i = p_{ii} + \max\left\{ \sum_{j\in N\setminus\{i\}} p_{ij}x_j \mid \sum_{j\in N\setminus\{i\}} \omega_j x_j \leq (c-\omega_i), x_j \in \{0,1\} \text{ for all } j \in N\setminus\{i\} \right\}. \tag{3}$$

A valid upper bound U^1_{cpt} for the QKP is the solution of the following problem:

$$\begin{aligned} \max \quad & \sum_{i\in N} \pi_i x_i \\ \text{s.t.} \quad & \sum_{i\in N} \omega_i x_i \leq c, \\ & x_i \in \{0,1\}, \quad \forall i \in N. \end{aligned} \tag{4}$$

By relaxing the integrality constraint in the computations of (3) and (4), we compute the bound U^{1*}_{cpt}, which is obtained in $\mathcal{O}(n^2)$ by solving n LP-relaxed knapsack problems for the computation of π_i for all $i \in N$, and one LP-relaxed knapsack problem for (4). Furthermore, Caprara et al. (1999) showed that the bound could be strengthened if the objective function is reformulated as follows:

$$\sum_{i\in N}\sum_{j\in N} p_{ij}x_ix_j = \sum_{i\in N}\sum_{j\in N} (p_{ij}+\lambda_{ij})x_ix_j, \tag{5}$$

for any skew-symmetric matrix $\Lambda = (\lambda_{ij})_{\substack{i\in N\\ j\in N}}$, i.e., $\lambda_{ij} = -\lambda_{ji}$. We denote $U^{1*}_{cpt}(\Lambda)$ as the corresponding bound with this new formulation. To obtain the tightest bound, we solve the Lagrangian dual problem $U^2_{cpt} = \min_{\{\lambda_{ij}:\lambda_{ij}=-\lambda_{ji}\}} U^{1*}_{cpt}(\Lambda)$. This can be solved by subgradient optimization, leading to the bound $\tilde{U}^2_{cpt}$ for a near-optimal matrix Λ.

The lower bound is obtained initially with the heuristic of Chaillou et al. (1983) combined with the local search "fill-up-and-exchange" (Gallo et al., 1980). Secondly, attempts to improve this bound are made by rounding down all variables from the solution obtained by solving the LP-relaxed knapsack problem in (4). Finally, Caprara et al. (1999) propose a variable reduction procedure that helps tighten the bound further. The idea is to fix a variable to its opposite value from the solution to the relaxed (4) and compute the upper bound. If the resulting bound is lower than the lower bound, we can remove the variable from the problem by fixing it to the appropriate value; otherwise, it is kept in the problem. The bounding algorithm of Caprara et al. (1999) iterates between the computation of an upper bound, a lower bound, and then variable reduction, stopping when no variable can be removed.

2.1.2 The Fomeni, Kaparis, and Letchford bounds

The bounding approach of Fomeni et al. (2022) relies on the integration of multiple sets of cutting planes into the reformulation of model (1) into a 0-1 LP model. Let us retain the variables x_i as previously defined and introduce variables y_{ij} for all $i, j \in N$ with $i < j$, such that $y_{ij} = x_i x_j$.

$$\begin{aligned}
\max \quad & \sum_{i \in N} p_{ii} x_i + \sum_{\substack{i,j \in N \\ i<j}} (p_{ij} + p_{ji})\, y_{ij} \\
\text{s.t.} \quad & \sum_{i \in N} \omega_i x_i \leq c, \\
& y_{ij} \leq x_i, && \forall i, j \in N, i \neq j, \\
& y_{ij} \geq x_i + x_j - 1, && \forall i, j \in N, i \neq j, \\
& x_i, y_{ij} \in \{0, 1\}, && \forall i, j \in N, i \neq j.
\end{aligned} \tag{6}$$

The first set of cutting planes is derived from the well-known valid inequalities of the knapsack polytope called lifted cover inequalities (Balas, 1975; Wolsey, 1975). Define $C \subset N$ as a cover set if $\sum_{i \in C} \omega_i > c$. A cover C is called minimal if no subset of C is a cover. This allows us to define the following valid inequalities of the knapsack polytope (Balas, 1975; Wolsey, 1975):

$$\sum_{i \in C} x_i + \sum_{i \in N \setminus C} \alpha_i x_i \leq |C| - 1. \tag{7}$$

Where α_i are lifting coefficients that are computed by solving a KP problem over the items in C by removing the weight of item i from the capacity. Balas (1975) showed that these coefficients could naively, but efficiently, be computed by the so-called extended cover set $E(C)$, where $E(C)$ denotes a subset of items in $N \setminus C$ that weight at least as much as the heaviest item in C. This results in an extended cover inequality if we set $\alpha_i = 1$ to all $i \in E(C)$ and $\alpha_i = 0$ to the remaining items $i \in N \setminus E(C) \cup C$.

Another set of cuts, called triangle inequalities, is derived from the binary quadratic programming literature (Padberg, 1989):

$$\begin{aligned}
x_i + x_j + x_k \leq y_{ij} + y_{ik} + y_{kj} + 1 \quad & \forall i, j, k \in N, i \neq j \neq k, \\
y_{ik} + y_{jk} \leq x_k + y_{ij} \quad & \forall i, j, k \in N, i \neq j \neq k.
\end{aligned} \tag{8}$$

Fomeni et al. (2022) also mention another set of inequalities called the cover tree (Johnson et al., 1993). Given a minimal cover C, we define K_C as the complete graph with vertices in C, and let T be the set of edges in the spanning tree in K_C. We define d_i for all $i \in C$ as the degree of the vertex in the tree. The cover tree inequality takes the following form:

$$\sum_{\{i,j\} \in T} y_{ij} \leq \sum_{i \in C} (d_i - 1) x_i. \tag{9}$$

Finally, Fomeni et al. (2022) strengthened their formulation by adding the reformulation-linearization technique (RLT) inequalities (Sherali and Adams, 1990). These inequalities are generated for both the extended cover inequalities and the capacity constraint. For instance, for the capacity constraint, this results in the following set of $2n$ constraints:

$$\begin{aligned} \sum_{i \in N \setminus \{j\}} \omega_i y_{ij} &\leq (c - \omega_j) x_j \quad \forall j \in N, \\ \sum_{i \in N \setminus \{j\}} \omega_i (x_i - y_{ij}) &\leq (1 - x_j) c \quad \forall j \in N. \end{aligned} \tag{10}$$

Regarding the lower bound, Fomeni et al. (2022) use two primal heuristic approaches. One is based on DP (Fomeni and Letchford, 2014), while the second uses the solution x^* obtained from the linear program (6) along with the inequalities mentioned before. The approach involves sorting the items in non-increasing order of their value in x^* and sequentially adding them into an empty knapsack if their weight does not exceed the residual capacity. Their bounding algorithm iterates between computing the upper bound by solving the linear program (6), computing the lower bound using this sorted approach, and then running separation routines to identify and add violated valid inequalities. The algorithm terminates when no further violated inequalities are detected.

2.2 Binary Decision Diagram Branch-and-Bound

The B&B algorithm for BDDs of Bergman et al. (2016) starts by adding the *root* node to a global priority queue P, which contains the nodes yet to be explored based on their attractiveness. For a given node $u \in P$, a restricted BDD $\underline{B}$ is constructed for the sub-problem rooted in u. Notably, as all paths in a restricted BDD are feasible solutions, the algorithm stores the longest path $v^*(\underline{B})$ of $\underline{B}$ plus the cost of the partial solution that led to node u. If this solution is better than the best-known solution (BKS) so far, the BKS is updated.

If $\underline{B}$ is exact, i.e., it completes the BDD rooted in u without restrictions, then the processing of u is concluded, and the algorithm moves on to explore another node. However, when $\underline{B}$ is not exact, a relaxed BDD $\overline{B}$ is compiled from u. This relaxed BDD serves two purposes: first, it provides a dual bound $v_u + v^*(\overline{B})$ on the profits achievable from exploring u, which is compared to the current BKS. This dual bound can be used to prune unpromising nodes. Second, even if no pruning is possible, the tree from the relaxed BDD is used to provide more nodes to queue P, called a *cutset*. When all nodes from P have been explored, an exact solution is obtained. If a time limit forbids exploring all the nodes in P, the method yields a heuristic solution, and no dual bound is available.

A cutset $\mathcal{S}$, in the context of BDDs, is defined as a subset of nodes from a relaxed BDD $\overline{B}$ such that any path from its *root* to its *terminal* node in $\overline{B}$ must go through at least one node in $\mathcal{S}$. A node u is said to be exact if all its incoming paths lead to the same state $s(u)$. An exact cutset of $\overline{B}$ is then composed entirely of exact nodes. Several cutsets can exist in a BDD. Below are the most significant ones.

- *Last Exact Layer* (LEL) means the set of nodes in the last not relaxed layer $\overline{B}$.
- *Frontier cutset* (FC) means all the nodes in $\overline{B}$ that are exact and have at least one relaxed direct child.

The DFS characteristic of the BDD B&B of Bergman et al. (2016) arises from the fact that the search is oriented by the attractiveness of a node in P, regardless of the depth or iteration of the B&B when the node was added to the pool.

3 BDDs for the QKP

Addressing a combinatorial optimization problem using BDDs requires a DP formulation (Bergman et al., 2014a). This section introduces a customized DP formulation for the QKP tailored for integration with BDDs. We will also discuss the compilation of an exact BDD using this formulation. We close this section with a numerical example.

3.1 DP model for the QKP

In Binary Quadratic Problems, certain elements, such as the value function computations, are common across many variants (Fomeni and Letchford, 2014; González et al., 2020). Hence, our proposed DP formulation for the QKP is inspired by the Quadratic Independent Set Problem formulation described by González et al. (2020). According to Bergman et al. (2016), the following components should be defined in a DP formulation for BDDs:

- S: The set of states, denoted as $\{S_0, \ldots, S_n\}$, where S_j represents a subset of all states at layer j in a BDD. The subset S_0 and S_n respectively represent the *root* state and the *terminal* state.
- Transition function $t_j : S_j \times \{0,1\} \longrightarrow S_{j+1}$ for $j = 1, \ldots, n$, which computes subsequent states based on decisions regarding the variable x_j in states S_j. This function ensures that transitions do not lead to infeasible states.
- Transition cost $h_j : S_j \times \{0,1\} \longrightarrow \mathbb{R}$ for $j = 1, \ldots, n$, determining the arc weights representing the profits obtained by fixing the value of x_j.
- Root value v_r, representing the constant value associated with the objective function at the *root* node.

Given a decision system of $n+1$ stages represented by BDDs, we determine, at each stage j, whether to add an item j to a given packing by setting the variable x_j to 0 or 1. To make this decision, we need information about the existing packing. Assuming a fixed variable ordering, we define the set $\mathcal{J} := \{k \in \{1, \ldots, j-1\} : x_k = 1\}$ as the partial packing at stage j, and $\gamma = c - \sum_{k \in \mathcal{J}} \omega_k$ as the residual capacity given the weight of the items in this packing. Thus, we define $\mathcal{I} := \{k \in \{j, \ldots, n\} : \omega_k \leq \gamma\}$ as the set of eligible items that could be added to packing $\mathcal{J}$. Additionally, to evaluate the marginal value of adding item k into packing $\mathcal{J}$, we must consider not only its linear profit but also the pairwise interaction with the items already in $\mathcal{J}$.

Through this approach, we can characterize the states S_j of a QKP represented in a BDD at stage j, with the tuple $\langle \mathcal{I}, q, \gamma \rangle$, where $\mathcal{I} \subseteq N$ is the set of eligible items based on the packing decisions made in stages $1, \ldots, j-1$ and $q = (q_1, q_2, \ldots, q_n)$ is a vector of n elements of the following sum:

$$q_i = p_{ii} + \sum_{k=0}^{j-1} 2p_{kj}x_k \quad \forall i \in N. \tag{11}$$

The value of x_k in Equation (11) corresponds to the assignments made in previous stages before reaching stage j. The *root* and the *terminal* states are respectively $s_r = \langle N, (p_{11}, \ldots, p_{nn}), c \rangle$ and $s_t = \langle \emptyset, \mathbf{0}, l \rangle$ with $l \in \left[0, \min_{k \in N} \omega_k\right[$. At the *root* state, all items are eligible, and their potential contribution is the linear profit associated with them; in the *terminal* state, no item is eligible, and no potential contribution exists. For the residual capacity at the *terminal* state, it could not exceed the lightest item; otherwise, it would mean that further packing is possible. The transition from states j to $j+1$, based on assigning a value to x_j, is computed with the following function:

$$t_j(\mathcal{I}, q, \gamma, x_j) := \begin{cases} \langle \mathcal{I} \setminus R_j, (q_1 + 2p_{1j}, \ldots, q_n + 2p_{nj}), \gamma - \omega_j \rangle & \text{if } x_j = 1 \\ \langle \mathcal{I} \setminus \{j\}, (q_1, \ldots, q_n), \gamma \rangle & \text{otherwise} \end{cases} \tag{12}$$

$$h_j(\mathcal{I}, q, \gamma, x_j) := q_j x_j, \tag{13}$$

where $R_j = \{j\} \cup \{k \in \mathcal{I} : \omega_k \leq \gamma - \omega_j\}$ is the set of items that have a weight lower than the residual capacity at the resulting state. Hence, R_j allows us to update the set $\mathcal{I}$ with the new eligible items. We also remove item j from the set of eligible items in both cases because we must include the information that we did decide on the assigned value variable x_j. Finally, the transition cost is simply the up-to-date q_j value if item j is selected or 0 otherwise. In some cases, when the set R_j contains more items than just j, the resulting state from the transition function does not contain all the items yet to be considered in its eligible set. Hence, to prevent falling into an infeasible state by evaluating the assignment of an item that does not exist in the eligible set, we wrap our transition function and cost into the following functions:

$$t_j^{'}(\mathcal{I}, q, \gamma, x_j) := \begin{cases} \langle \mathcal{I}, q, \gamma \rangle & \text{if } j \notin \mathcal{I} \\ t_j(\mathcal{I}, q, \gamma, x_j) & \text{otherwise} \end{cases} \tag{14}$$

$$h_j^{'}(\mathcal{I}, q, \gamma, x_j) := \begin{cases} 0 & \text{if } j \notin \mathcal{I} \\ h_j(\mathcal{I}, q, \gamma, x_j) & \text{otherwise.} \end{cases} \tag{15}$$

Lastly we define the set of decisions $\mathcal{F}(C) := \{0\} \cup \{\mathbb{I}(C)\}$, where $\mathbb{I}(C)$ equals to 1 if condition C holds and 0 otherwise. Considering these presented elements, we can articulate the following Bellman equations that are solved with the optimal solution of the QKP.

$$\begin{aligned} V_j(\mathcal{I}, q, \gamma) &= \max_{d \in \mathcal{F}(j \in \mathcal{I})} (h_j^{'}(\mathcal{I}, q, \gamma, d) + V_{j+1}(t_j^{'}(\mathcal{I}, q, \gamma, d))) \\ V_{n+1}(\mathcal{I}, q, \gamma) &= 0. \end{aligned} \tag{16}$$

The states at layer $n+1$ are terminal; hence, their value equals 0. Starting from the *root* state s_r, $V_1(s_r)$ yields an optimal solution to the QKP. The root value v_r is initially set to zero as there is no accumulated profit. Its purpose will be clearer in Section 6 when we discuss BDD B&B.

3.2 Long transitions

It is common in BDDs literature to use long transitions between states (Bergman et al., 2016). However, to our knowledge, it has never been applied in combinatorial optimization. The concept of a long transition is to combine multiple decisions to transit between two adjacent states. This has the advantage of reducing the state space size, as several states would be explicitly combined. This section introduces two scenarios where long transitions are particularly relevant when certain fundamental QKP assumptions are unmet. More specifically, we define particular states where the transition to the *terminal* state is straightforward. For all states, we can formulate a subproblem for the items in $\mathcal{I}$ with a maximum capacity of γ. To warrant an efficient exploration of these subproblems, they should satisfy the following properties.

Property 1. The maximum capacity of the knapsack γ must be strictly less than the sum of the weights of all items in $\mathcal{I}$.

Property 2. The maximum number of items m that could fit into the knapsack must be at least 2.

The first property highlights a crucial concept in the QKP, indicating that it is impossible to select all items. This is confirmed by checking whether the maximum number of items m that could fit into the knapsack is equal to the number of items in the eligible set $|\mathcal{I}|$. If this condition is true, the exact solution to the problem can be easily obtained by assigning a value of 1 to all the remaining decision variables. The second property prevents a situation where selecting only one item is the sole possibility. Hence, states in the BDD graph that do not satisfy at least one of these properties are immediately linked to the *terminal* state. We introduce a modified transition function t_j^+.

$$t_j^+(\mathcal{I}, q, \gamma, x_j) := \begin{cases} (\emptyset, \mathbf{0}, 0) & \text{if } m = 1 \text{ or } m = |\mathcal{I}| \\ t_j^{'}(\mathcal{I}, q, \gamma, x_j) & \text{otherwise,} \end{cases} \tag{17}$$

where m represents the maximum number of items that can fit into the knapsack, computed by arranging the items in $\mathcal{I}$ in non-decreasing order of their weights and sequentially placing them into the knapsack until the capacity is attained. We propose the modified function h_j^+ for the transition cost.

$$h_j^+(\mathcal{I}, q, \gamma, x_j) := \begin{cases} \max\limits_{i \in \mathcal{I}} q_i & \text{if } m = 1 \\ \sum\limits_{i \in \mathcal{I}} q_i + \sum\limits_{\substack{i,k \in \mathcal{I} \\ i \neq k}} p_{ik} & \text{if } m = |\mathcal{I}| \\ h_j^{'}(\mathcal{I}, q, \gamma, x_j) & \text{otherwise.} \end{cases} \tag{18}$$

In the first case, the knapsack can only fit one item among all eligible ones. Thus, the best solution is to select the item with the highest q_i. The second case tests if the total weight is less than or equal to the residual capacity, which leads to computing the overall profit of selecting all items. The linear value of the items in $\mathcal{I}$ becomes equal to the values in q as it considers the linear profit of items and their pairwise interaction with items $\mathcal{J}$. If none of these tests is true, i.e., Properties 1 and 2 are verified, then we return to the usual transition function from the previous section.

3.3 BDD Compilation

Equipped with the DP formulation introduced above, we outline the BDD construction in Algorithm 1. Let s_u denote the state at node u, $\mathcal{I}_u$ the set of eligible items in state s_u, and r and t respectively the *root* and *terminal* nodes. Algorithm 1 begins by creating a *root* node and placing it in the initial layer l_0. For each node u in the current layer l_i, the algorithm explores possible binary decisions d that are allowed by the set $\mathcal{F}(j \in \mathcal{I}$ **and** $m \neq 1$ **and** $m \neq |\mathcal{I}|)$. We limit the number of allowed decisions to one (instead of two) when one of the conditions in $\mathcal{F}$ is verified, as in those cases the actions to take are trivial and do not depend on the set of binary decisions as described in Equations (14), (15), (17), and (18). For each decision, it creates a new node u_d with a corresponding state s_d determined by the transition function. An arc $a_d(u)$ from node u to u_d is established with a weight $v(a_d(u))$ computed using the transition cost function. The new node u_d is then added to the set of nodes for the next layer l_{i+1}. This iterative process continues until all $n+1$ layers are processed, ensuring the sequential construction of the BDD.

Algorithm 1 Exact BDD

1: Set $W \longleftarrow +\infty$ for an exact BDD; Set W for a restricted or relaxed BDD
2: Create *root* node r and let $l_0 = \{r\}$
3: **for** $i \in N$ **do**
4: **if** $|l_i| > W$ **then**
5: $C_i \leftarrow$ cluster_nodes(l_i), $l_i \leftarrow l_i\{\oplus n_i : \forall n_i \in C_i\}$
6: **end if**
7: $j \leftarrow select\_var(l_i)$
8: **for** $u \in l_i$ **and** $d \in \mathcal{F}(j \in \mathcal{I}$ **and** $m \neq 1$ **and** $m \neq |\mathcal{I}|)$ **do**
9: $s_d \leftarrow t_j^+(s_u, d)$ and create a new node u_d with state s_d
10: $a_d(u) \leftarrow (u, u_d)$, $v(a_d(u)) \leftarrow h_j^+(s_u, d)$, $l_{i+1} \leftarrow l_{i+1} \cup \{u_d\}$
11: **end for**
12: **end for**
13: **Return** the longest path between the root r and the terminal node t

The condition in line 4 of the algorithm limits the layer's size to a maximum of W nodes. For the case of an exact compilation of BDD, we use $W = \infty$. The two operations in line 5 concern approximate BDDs, which we will discuss in the next sections. The function *cluster_nodes*$(\cdot)$ clusters the nodes in the argument, and the operator $\oplus$ aggregates the nodes in each cluster to produce a reduced set of aggregated nodes. The operation *select_var*$(\cdot)$ in line 7 allows for dynamic variable ordering. While several ordering approaches exist in the literature (Bergman et al., 2012), our experiments showed that the dynamic approach below leads to a better performance:

$$select\_var(l_i) := \arg\max \{q_j \mid j \in \mathcal{I}_u, u \in l_i\},$$

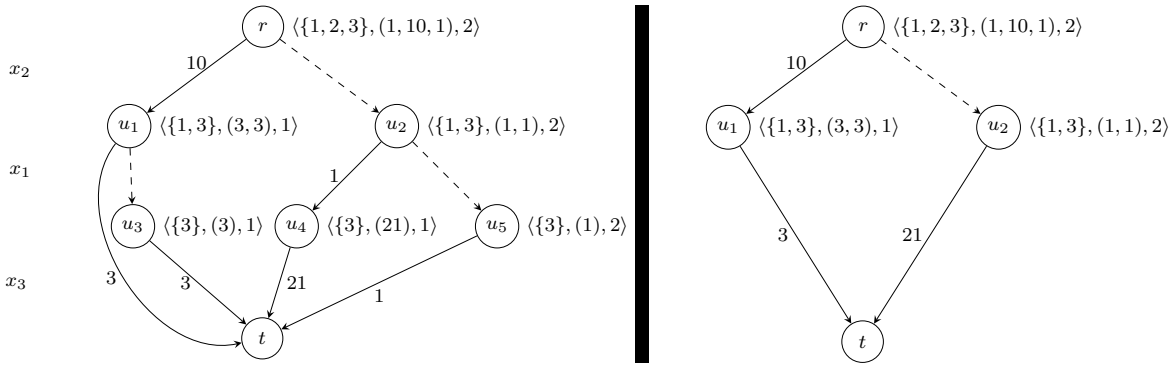
where $\mathcal{I}_u$ is the set of eligible items in node u. Finally, after the BDD is compiled, Algorithm 1 returns the longest path between the *root* and *terminal* node.

3.4 Numerical example

We introduce a simple example on a QKP instance to illustrate the BDD tree. We consider the following instance with $n = 3$ items, weights $\omega = (1, 1, 1)$, capacity $c = 2$, and the profit matrix P:

$$P = \begin{bmatrix} 1 & 1 & 10 \\ 1 & 10 & 1 \\ 10 & 1 & 1 \end{bmatrix}.$$

Figure 1a shows the construction of an exact BDD using the formulation from Section 3.1, and Figure 1b shows the result when the long transitions from Section 3.2 are considered. The main difference between both BDDs is that by using long transitions, we reduce the number of nodes by half. Furthermore, irrelevant paths (potential solutions), i.e., $r \to u_2 \to u_5 \to t$, are not even considered with long transitions. At node u_1, the long transition is activated because Property 2 is not verified. Hence, it is automatically linked to the *terminal* node with an arc weight of $\max_{i \in \mathcal{I}} q_i = 3$ at the state $s(u_1)$. For node u_2, Property 1 is not verified; thus, the best solution would be to include all items with an arc weight of $\sum_{i \in \mathcal{I}} q_i + \sum_{\substack{i,k \in \mathcal{I} \\ i \neq k}} p_{ik} = 21$ at state $s(u_2)$.



(a) BDD without long transitions **(b) BDD with long transitions**

Figure 1: Exact BDD

4 Restricted BDD

Restricted BDDs are designed such that whenever a layer l_i exceeds a maximum width size W, a subset of its nodes is selected and discarded. In this way, the resulting longest path in the BDD is a valid *lower bound* to the optimal solution. This is because all the paths in the restricted BDD still encode feasible solutions to the problem, but they may not conserve the optimal ones.

Hence, restricted BDDs must be designed to navigate scenarios where the explored parts of the solution space demand a more efficient representation. Intuitively, this can be achieved by removing nodes u that have the shortest longest path $v^*(\underline{B}_{ru})$ to the *root* node r in the restricted BDD $\underline{B}$ (Bergman et al., 2014c). However, in the case of BQPs, this is not enough as it does not take into account the potential profits from the interaction with items in future iterations. Fennich et al. (2024) proposed an approach to evaluate states of QKP solution in a DP framework, where the future profits are considered. In this section, we present this approach for node selection. The idea is based on a

fast calculation of a local lower bound to the subproblem encoded in each BDD node, denoting u_{lb}. The subproblem encoded in a BDD state is also a QKP that is formulated as follows.

$$\begin{aligned} \max \quad & \sum_{i\in\mathcal{I}} q_i y_i + \sum_{\substack{i,j\in\mathcal{I} \\ j>i}} 2p_{ij} y_i y_j \\ \text{s.t.} \quad & \sum_{i\in\mathcal{I}} \omega_i y_i \leq \gamma, \\ & x_i \in \{0,1\}, i \in \mathcal{I}. \end{aligned} \tag{19}$$

In this subproblem, we define a QKP on the set of items in $\mathcal{I}$, where the capacity is the residual capacity γ at node u. The quadratic profits between the items remain unchanged from the main problem, but their linear profits are updated to the current values q_i. These updated linear profits reflect the standalone profit from adding item i, which includes its linear profit plus the pairwise gains with the items already selected in the previous layers. This subproblem formulation has the potential to give a future estimation of the profit if solved exactly. Extensive computations by Fennich et al. (2024) show that heuristically solving problem (19) with the algorithm of Chaillou et al. (1983) from multiple DP states is sufficient to reach good-quality results for the QKP. Furthermore, they proposed a very fast local search called remove-and-fill-up that checks if the solution of Chaillou et al. (1983) could be improved. The idea of this local search procedure is to remove an item k from the solution, freeing up residual capacity. Items not yet in the solution with weights less than or equal to residual capacity are identified and used to define a new QKP. The new QKP is formulated for these items, considering their linear and quadratic profit contributions with the items still in the knapsack. This subproblem is solved using the heuristic of Chaillou et al. (1983) to find a subset that maximizes the profit. If this new subset improves the overall solution, it replaces item k. This process is repeated for different items in the solution.

We denote Q_u, the subproblem encoded at node u, and $z(Q_u)$ is its solution. We define $u_{lb} = v^*(\underline{B}_{ru}) + z(Q_u)$ as the selection criteria for width restriction. The selection process for restricted BDDs is to set all $|l_i| - W$ nodes with the lowest u_{lb} values and return a set of all nodes. Then, in the aggregation operation $\oplus$ in line 5 of Algorithm 1, these identified nodes are deleted, and the remaining ones are returned.

5 Relaxed BDD

Relaxation in BDDs constitutes an extension of the problem as unlike restricted BDDs that underapproximate the solution space, relaxed BDDs overapproximate it when it comes to limiting the layer's width. In other words, the aggregation phase in Algorithm 1 does not *remove* states, but *merges* them, which may lead to an infeasible state, resulting in a valid upper bound. This is done by proposing a merging operation $\oplus$ between the states of the nodes for the aggregation process. The resulting state should contain the information from both input states. For the QKP, we propose the following merge operation $w = \oplus(\{u, v\})$ where the state $s(w)$ of the merged node w becomes $\mathcal{I}_w := \mathcal{I}_u \cup \mathcal{I}_v$, $q_w := \max(q_u, q_v)$ with $q_{w_i} = \max(q_{u_i}, q_{v_i})$ and $\gamma_w = \max(\gamma_u, \gamma_v)$. This is a valid merge as the new state considers all the items from both eligibility sets; hence, no potential assignment is lost. Furthermore, the residual capacity and the potential profit are maximized such that the resulting paths from a merged node would lead to at least the optimal value, hence the dual-bound nature of this solution. Figure 2 shows a possible relaxation to the BDD in Figure 1a for $W = 2$.

In Figure 2, the nodes u_4 and u_5 are merged. The resulting state maximizes the residual capacity and allows as much space in the packing as possible for future item selection. At the level of the profit vector, the maximum potential value of item 3 is considered which, in this case, conserves the optimal path to the *terminal* node within the feasible space.

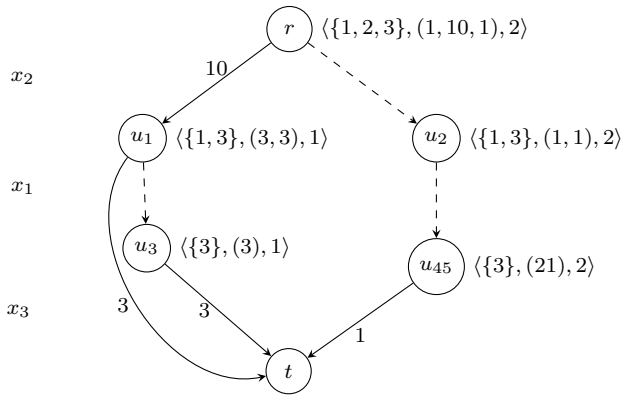


Figure 2: Relaxed BDD of Figure 1a

The selection of nodes for merging is crucial in ensuring tight bounds in relaxed BDDs. A greedy strategy is to merge nodes with similar states. Bergman and Cire (2017) introduced an IP model to compute the tightest relaxed BDD for a given W. However, their approach was primarily evaluated on the knapsack problem and demonstrated effectiveness only on small-sized instances. In what follows, we propose a heuristic method designed to achieve similar objectives if one can define a similarity measure.

5.1 Node clustering

In the last decade, MSTs have been widely used for data clustering in different areas (Grygorash et al., 2006). Constructing an MST from a distance matrix that encapsulates pairwise similarities between states provides a hierarchical representation of relationships within the state space. Subsequently, clusters are delineated by strategically cutting the branches of the MST, yielding a cohesive partitioning of the states. This approach offers insights into the intrinsic structure of the state space and facilitates meaningful interpretations of inter-state relationships. In this section, we propose a methodology for clustering nodes using MSTs for our relaxed BDDs.

Let us consider the graph $G = (l_i, E)$ that contains all nodes from the layer l_i in a relaxed BDD. The weighted edges $e \in E$ represent relationships between those nodes, and their weights encode a measurement of similarity (discussed in the next section). We assume for now that G is fully connected, but this does not always hold for QKPs as seen next. Equipped with G, we can run a variant of Kruskal algorithm (Cormen et al., 2009) that returns the set of connected components to be merged in the relaxed BDD. The Kruskal algorithm should stop when the number of connected components in the MST equals the maximum layer width W. This variant of the Kruskal algorithm begins as usual by sorting the edges of the input graph in a non-decreasing order of weight. The set of connected components $\mathcal{T}$ is initialized, with each vertex defining an initial component, i.e., $|\mathcal{T}| = |l_i|$. As the algorithm progresses through the sorted edges, it evaluates whether merging two components would violate the acyclic property of a tree. If a merging operation is permissible, the corresponding edge is "added" to the MST, and the two components are merged into a single connected component. This operation decreases the number of connected components, and the algorithm stops when the required number of components is met or when a full MST is computed. A detailed pseudocode of this procedure is provided in Appendix A.

5.2 Distance graph to measure similarity

Constructing a suitable distance graph is crucial to the success of our MST clustering. In this section, we propose a distance graph for the states of a QKP, which allows us to measure similarity between nodes and thus enables efficient node clustering to be merged in relaxed BDDs.

First, recall $u_{lb} = v^*(\overline{B}_{ru}) + z(Q_u)$ as the local lower bound of node u in a relaxed BDD $\overline{B}$. This information helps guide the clustering to merge nodes potentially leading to the worst solutions. The reason is to reduce the overapproximation (relaxation) impact, as the solution we obtain is that of the longest path. We define the set $K \subset l_i$ that contains $|l_i| - W$ nodes with the lowest u_{lb}. Furthermore, in preliminary experiments, we noticed that to ensure the tightness of the bound for the QKP, it is crucial to limit the merge only to nodes containing the same maximum number of items m to be added (sorting the items in l_i in non-decreasing order and adding the first ones until the residual capacity is filled). Hence, we define the distance graph for the QKP as $G = (l_i, E)$ where $E = \{(u_1, u_2) | u_1 \in K, u_2 \in l_i, m_{u_1} = m_{u_2}\}$, and m_u is the number of items from the eligible set $\mathcal{I}_u$ that could be included given the residual capacity γ_u. This formulation may lead to a sparse graph, consequently resulting in insufficient edges to reduce the number of connected components to the desired W if $|E| < |l_i| - W - 2$. However, this allows us not only to consider resource allocation during layer reduction but also the complexity of the state space. This changes the definition of W from the maximum allowed width to the preferred width, if feasible. Furthermore, our experiments showed that this potential expansion of layer width is manageable, reaching only a few hundred in the worst cases.

Regarding the edges weight, we compute a distance measure that considers the difference in the attractiveness of outcomes between states and the extent of overestimation computed by the merge. Therefore, we propose $D(u_1, u_2) := |u_{1lb} - u_{2lb}| + \lambda |\gamma_{u_1} - \gamma_{u_2}|$ as the distance between nodes u_1 and u_2, where λ is a scaling factor intended to emphasize the importance of the capacity constraint. We found it crucial to restrict the merging of nodes to those with very similar residual capacities; thus, we set $\lambda = 1 \times 10^6$.

6 Our Binary Decision Diagram Branch-and-Bound

Bergman et al. (2016) introduced a DFS BDD-based B&B algorithm that uses information computed from approximate BDDs to improve the search process within an exact BDD, terminating upon finding an optimal solution. However, obtaining an optimal solution for complex problems can take a significant amount of time. The DFS nature of the search makes it such that the dual bound available during the search can be very weak (i.e., that of the very first iteration, if the nodes added to the pool during the first iteration are yet to be explored). Consequently, if the search terminates due to a time limit, the quality of the obtained solution cannot be evaluated efficiently.

In this section, we propose a BFS BDD-based B&B algorithm designed to extract tight bounds. Furthermore, we also discuss the parallelization of this search scheme.

The definition of exact cutsets implies that they act as a frontier before which the relaxed BDD $\overline{B}$ and its exact counterpart B have not diverged. Hence, the information on the longest path in B from a tree rooted in r could always be reached with the following equation:

$$v^*(Sol(B)) = \max_{u \in \mathcal{S}} (v^*(Sol_u(B))) \tag{20}$$

where $Sol_u(B)$ is a restriction on the solution space that crosses node u, and $v^*(Sol_u(B)) = v^*(B_{ru}) + v^*(B_{ut})$. Equation (20) states that at least one path is optimal among all paths that cross $\mathcal{S}$. This implies that one can tighten the upper bounds in a relaxed BDD by fully exploring each node of an exact cutset. This idea is demonstrated in the lemmas below. We introduce $\mathcal{S}_u$, the exact cutset returned from $\overline{B}$ when it is rooted at node u.

Lemma 1. If $\mathcal{S}$ is an exact cutset of $\overline{B}$ then $\max_{u \in \mathcal{S}} (v^*(Sol_u(\overline{B})))$ is a valid upper bound on $v^*(Sol(B))$

Proof. Since the nodes $u \in \mathcal{S}$ are exact, then $v^*(Sol_u(\overline{B})) = v^*(B_{ru}) + v^*(\overline{B}_{ut})$ because there is no loss of information in the path from the root r to u. Hence, by the definition of Equation (20) we can state $v^*(\overline{B}_{ut}) \geq v^*(B_{ut}) \iff \max_{u \in \mathcal{S}} (v^*(Sol_u(\overline{B}))) \geq v^*(Sol(B))$. □

Lemma 2. If $\mathcal{S}$ is an exact cutset of $\overline{B}$ then $\bigcup_{u \in \mathcal{S}} \mathcal{S}_u$ is also an exact cutset of $\overline{B}$.

Proof. The proof is trivial as $\bigcup_{u \in \mathcal{S}} \mathcal{S}_u$ also acts as a complete frontier before which the relaxed BDD $\overline{B}$ and its exact counterpart B have not diverged. □

We propose a BFS B&B in Algorithm 2 that iterate over multiple exact cutsets of the BDD tree. The goal is to explore the nodes of full cutsets, yielding tightened upper bounds throughout the search. While our approach, similarly to the one of Bergman et al. (2016), requires the usage of a priority queue P to feed the exploration process, we propose that new cutsets are added to a temporary set $\mathcal{C}$, which feeds P once P is empty. This is fully explained next.

Algorithm 2 BFS BDD-based branch-and-bound

```
Create root node r, C ← {r}
Set z ← −∞, z̄ ← ∞
while C ≠ ∅ do
    P ← {u|u ∈ C, u_lub > z}
    z̄ ← max(u_lb|u ∈ P),   C ← ∅
    for u ∈ P do
        Explore node u with Algorithm 3
    end for
end while
return z, z̄
```

Our proposed algorithm starts by creating a *root* node r and establishing the set of cutsets $\mathcal{C}$ initialized with a singleton element r. Then, the lower bound and upper bound are initialized respectively $\underline{z} = -\infty$ and $\overline{z} = \infty$. The main loop of the algorithm persists until there is no remaining node to be explored in $\mathcal{C}$. Within each iteration, the algorithm feeds nodes from the exact cutset $\mathcal{C}$ to the global problem into the priority queue P. These nodes must possess a local upper bound u_{lub} exceeding the current lower bound $\underline{z}$ where u_{lub} is a fast local upper bound on node u, proposed by Gillard et al. (2021). It is computed with a bottom-up approach, cumulating the arc weights in the paths between the *terminal* node t and u in the relaxed BDD $\overline{B}$. This is practically done by inserting the local upper bound calculation procedure in Appendix A at the end of Algorithm 1 when it is a relaxed BDD compilation, with the *terminal* node and the weights of its incoming edges being the arguments. This local upper bound is initially equal to zero for all nodes. The algorithm then clears the cutset $\mathcal{C}$ and updates the global upper bound $\overline{z}$ by considering the maximum u_{lb} over all nodes in P.

Algorithm 2 explores the nodes from P using the procedure described in Algorithm 3, following their order of priority. For a given u, we first compute a restricted BDD $\underline{B}$ and update the current best-known solution $\underline{z}$ if a better one is found by adding the best value at the current node v_u to the value of longest path in $\underline{B}$. If the restricted BDD $\underline{B}$ is not exact, the algorithm then computes a relaxed BDD $\overline{B}$. If the relaxed BDD also yields a better solution, the algorithm updates the set of cutsets $\mathcal{C}$ by including an exact cutset of $\overline{B}$.

Unlike the approach of Bergman et al. (2016) mentioned earlier where the exact cutset from exploration of node u is directly fed to the priority queue P, our approach stores it in the set of exact cutsets $\mathcal{C}$ instead of the queue P. Furthermore Algorithm 2 waits for all the nodes in P to be explored to start the next iteration to ensure that the construction of a new valid exact cutset $\mathcal{C}$ to the main problem is complete. If the latest is empty, at the next iteration, the algorithm stops with a proof of optimality (Bergman et al., 2016); otherwise, it reiterates with the new set of extracted nodes.

The BDD B&B approach is easily parallelizable, as the exploration of nodes is independent. This offers the advantage of achieving tighter upper bounds by accelerating the exploration of a complete exact cutset. Moreover, this may lead to faster optimality in both BFS and DFS BDD B&B algorithms (Bergman et al., 2014b).

Algorithm 3 Exact nodes exploration procedure

procedure EXPLORE(u) ▷ args: root node u
  $\underline{B} \leftarrow Restricted(u)$
  if $v_u + v^*(\underline{B}) > \underline{z}$ **then**
    $\underline{z} \leftarrow v_u + v^*(\underline{B})$
  end if
  if $\underline{B}$ is not exact **then**
    $\overline{B} \leftarrow Relaxed(u)$
    if $v_u + v^*(\overline{B}) > \underline{z}$ **then**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{S}$ where $\mathcal{S}$ is an exact cutset of $\overline{B}$
    end if
  end if
  if $\overline{z} = \underline{z}$ **then**
    stop the search as optimality is proven
  end if
end procedure

Hence, we approach the parallelization as follows. We use a thread pool and enqueue the task of node exploration at line 7 of Algorithm 2 into this thread pool according to its priority in P. The master thread waits for all nodes in P to be processed before trying to make another iteration in the main loop. Finally, even if the optimality is not reached, one can stop the process according to a computational time limit and evaluate the quality of its feasible solution. This is helpful as it allows a comparison between the bounds obtained with the algorithm of Caprara et al. (1999), Fomeni et al. (2022), and our approach, as we will see in the computational results section next.

7 Computational results

This section presents the results and analyses of the computational experiments conducted to test the efficiency of our proposed bounding algorithm. All the discussed algorithms are coded in C++ and executed on machines equipped with 2 × Intel E5-2683 v4 Broadwell @ 2.1GHz with 32 GB of RAM. We limit the computation of our experiments to 1 hour. We set the required maximum width W to 100 for the restricted BDDsd and 10 for the relaxed BDDs. Our choice of the parameter W is motivated by the trade-off between solution quality and resource consumption after a preliminary tuning phase. Furthermore, larger layer widths lead to larger cutsets; hence, it is relevant to consider a smaller width when it comes to relaxed BDDs, as it will tighten the upper bound faster because of the fewer nodes that need to be explored. Meanwhile, for restricted BDDs, larger widths may lead to better primal solutions. This explains our choice to have different widths for each BDD type.

Section 7.1 describes the test instances. In Section 7.2, we report the qualities of the bounds for our methods, which are compared against the bounds of Caprara et al. (1999) and Fomeni et al. (2022), which were also reimplemented. Finally, in Section 7.3, we report a computational experiment comparing the BFS B&B that we propose with the DFS B&B of Bergman et al. (2016).

7.1 Test instances

We now describe the benchmark instances used to evaluate the performance of our approach, encompassing both the longstanding standard instances from Gallo et al. (1980), the instances derived from the dispersion and the densest subgraph problem (Pisinger et al., 2007), and the recent Hidden Clique instances introduced by Schauer (2016).

Standard QKP Instances: the standard QKP instances, introduced by Gallo et al. (1980), have been used for nearly four decades. Each item's weight is a random integer between 1 and 100, and the knapsack capacity is a randomly chosen integer between 50 and the total weights' sum. The profit matrix's sparsity is generated for each case $\{25\%, 50\%, 75\%, 100\%\}$. We generated 240 instances for various values of n and sparsity probabilities.

Dispersion Problem Instances: the dispersion problem is a combinatorial optimization problem where k facilities should be assigned to locations while maximizing the distance between the chosen locations. The problem serves as another set of instances by formulating it as a special case of the QKP (Pisinger et al., 2007). Eight variants were generated according to the distribution used to generate the distance matrix, including Geometrical (GEO), Weighted Geometrical (WGEO), Exponential (EXPO), and Uniform distributions (RAN). Additionally, we generated knapsack versions of these variants (KP-{EXPO, GEO, WGEO, RAN}) where each location has a random weight in $\{1, \ldots, 100\}$. A total of 400 instances were generated for this experiment.

Densest k Subgraph Instances: the densest k subgraph problem, a well-studied optimization problem, was employed in four variants: DSUB25, DSUB50, DSUB75, and DSUB90, with varying sparsity levels in the profit matrix. The aim is that given a graph, we need to find the densest subgraph with k nodes, which can be formulated as a QKP (Pisinger et al., 2007). A total of 200 instances for different combinations of size and sparsity were generated.

Hidden Clique Instances: the hidden clique instances involve finding a hidden clique in a random graph. These instances were introduced by Schauer (2016) and are known to be particularly challenging (Alon et al., 2011). A total of 200 instances were generated for this experiment.

This diverse set of instances covers a broad spectrum of QKP variants, comprehensively assessing our approach's performance.

7.2 Bounds for the QKP

This section presents the computational results over all the instances just described. We compute both the lower and the upper bounds using the algorithms reviewed in Section 2.1, as well as our parallelized BFS BDD B&B using both LEL and FC. We use 24 cores for parallelization and allocate 1 GB of RAM for each. For each algorithm, we report the average results of the following three metrics:

- Gap: the average percentage gap between the lower bound (LB) and the upper bound (UB), computed as $\frac{UB - LB}{UB}$.
- Deviation (Δ): the average percentage gap between the lower bound and the best-known solution BKS from any of the algorithms. This gap is computed as $\frac{(BKS - LB)}{BKS}$. Zero values indicate that the current algorithm produced the BKS.
- Time: the average runtime required for the execution of the algorithm.

The combination of Gap and Δ allows us to obtain useful insights into the quality of both gaps. If $Gap_1 \leq Gap_2$ and $\Delta_1 \leq \Delta_2$, then algorithm 1 has better performance with better lower and upper bounds; if $Gap_1 \leq Gap_2$ and $\Delta_1 \geq \Delta_2$, then algorithm 1 has a less attractive lower bound but a better upper bound; if $Gap_1 \geq Gap_2$ and $\Delta_1 \leq \Delta_2$, then algorithm 1 has a better lower bound but a less attractive upper bound.

7.2.1 Results for standard instances

We report in Table 1 the results on the standard QKP instances randomly generated with the procedure of Gallo et al. (1980). In this category of instances, we can observe the best feasible solution is obtained with the algorithm of Fomeni et al. (2022) with an average deviation Δ from the BKS of 0.01%. However, when it comes to the Gap, Fomeni et al. (2022) have the least interesting results with an average over 38%, meaning that their approach reports the worst upper bounds. On the other hand, Caprara et al. (1999) and both BDD approaches lead to similar performances on average for both the gap and the deviation, with FC being the dominating cutset for BDDs in these instances. For the lower bound, both methods report interesting deviations of less than 0.5% for most instances and an average of 0.2%. For the upper bound, we observe that the BDD approach leads to the best bounds

for small instances with up to 200 items, but it fails to remain competitive above this size limit. This can be explained by the resulting quality of the lower bounds for those instances due to the width limit of 100 for the restricted BDD. We observed that to obtain better results, the required increase in the width limit is significant, which leads to more resource consumption; hence, the trade-off is not justified. However, the average difference between the two deviations Δ of our BFS BDD B&B and that of Caprara et al. (1999) is 0.5% and its median is 2%. This means that they are tight compared to the difference between the upper bound of Caprara et al. (1999) and the one of Fomeni et al. (2022).

size	density	Caprara et al. (1999)			Fomeni et al. (2022)			BDD (LEL)			BDD (FC)		
		Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)
50	25	18.07	1.01	4.30	0.28	0.20	3.15	0.00	0.00	31.52	0.00	0.00	2.23
	50	17.01	0.29	6.11	0.54	0.00	3.34	0.00	0.00	5.85	0.00	0.00	0.96
	75	17.05	0.25	7.78	0.20	0.00	5.76	0.00	0.00	4.67	0.00	0.00	1.18
	100	11.50	0.38	7.20	0.43	0.00	5.14	0.00	0.00	2.32	0.00	0.00	0.69
100	25	21.83	0.49	16.91	29.59	0.03	14.52	20.74	0.22	2900.36	14.41	0.22	2889.03
	50	17.34	0.17	18.63	30.77	0.00	39.27	13.70	0.05	2505.56	8.16	0.04	1460.40
	75	21.21	0.16	21.88	30.63	0.00	29.16	22.98	0.22	3609.52	14.27	0.04	2899.83
	100	10.91	0.06	22.46	24.87	0.00	15.98	9.11	0.00	2716.68	0.00	0.00	188.50
150	25	5.98	0.14	33.11	9.60	0.00	197.51	5.74	0.12	1094.08	4.71	0.12	903.96
	50	30.40	0.53	41.72	31.90	0.00	433.63	31.63	0.49	3605.51	26.01	0.49	2960.73
	75	19.61	0.17	46.32	34.37	0.00	18.30	22.33	0.43	3279.34	18.13	0.43	2887.40
	100	9.18	0.07	56.83	7.73	0.00	66.25	8.17	0.02	3162.24	2.81	0.00	1456.52
200	25	31.50	0.43	63.56	18.93	0.00	812.17	31.11	0.13	3604.34	28.38	0.13	2909.91
	50	36.45	0.31	77.70	78.05	0.04	1867.86	38.19	0.22	3605.49	33.66	0.22	3603.81
	75	22.69	0.30	85.05	68.93	0.02	46.70	29.26	0.26	3603.61	24.62	0.26	3603.92
	100	12.55	0.36	91.76	45.28	0.00	54.06	21.30	0.66	3603.26	16.32	0.66	3604.03
250	25	18.64	0.17	93.32	33.12	0.00	2680.61	23.16	0.30	3606.77	20.90	0.30	3606.08
	50	15.47	0.17	101.48	28.57	0.00	984.90	19.49	0.21	3606.21	17.10	0.21	3605.52
	75	19.64	0.16	133.65	50.66	0.00	1389.71	25.96	0.45	3607.84	23.08	0.45	3606.36
	100	12.80	0.04	155.29	43.85	0.00	145.84	23.67	0.55	3607.42	19.48	0.55	3606.29
300	25	17.96	0.20	152.85	33.17	0.00	2858.90	19.04	0.07	2891.31	15.85	0.11	2176.31
	50	27.23	0.20	172.41	54.49	0.00	2701.74	31.45	0.28	3610.57	29.27	0.28	3610.26
	75	19.20	0.07	196.46	14.52	0.00	434.27	26.07	0.05	3609.50	22.70	0.05	3608.97
	100	13.00	0.17	231.69	51.22	0.00	171.80	24.04	0.22	3611.03	22.19	0.22	3610.79
350	25	27.27	0.13	191.36	44.20	0.00	2952.62	29.47	0.27	3618.17	28.19	0.27	3616.75
	50	31.93	0.10	247.17	58.09	0.00	3107.40	35.48	0.41	3621.32	34.58	0.41	3620.08
	75	22.86	0.09	290.41	65.09	0.00	1268.44	31.27	0.38	3616.32	28.86	0.38	3616.56
	100	10.80	0.04	303.21	47.82	0.00	263.87	19.16	0.24	3618.44	17.47	0.24	3618.18
400	25	34.64	0.17	271.55	54.58	0.00	3607.30	35.35	0.20	3628.83	34.22	0.20	3627.55
	50	17.38	0.03	278.21	28.42	0.00	1017.86	22.07	0.30	3623.79	20.82	0.30	3623.63
	75	24.50	0.24	409.19	74.83	0.00	1238.26	34.47	0.10	3628.61	32.91	0.10	3628.03
	100	9.85	0.08	407.95	35.91	0.00	2498.12	15.94	0.08	3626.49	14.88	0.08	3624.58
450	25	20.57	0.40	341.49	36.45	0.00	3608.77	24.84	0.24	3630.93	23.85	0.24	3629.74
	50	17.52	0.16	377.56	23.61	0.00	2576.66	21.55	0.32	3623.79	19.87	0.32	3195.12
	75	22.42	0.08	536.01	61.02	0.00	2383.15	30.43	0.15	3644.18	29.25	0.15	3642.65
	100	9.52	0.05	539.57	34.87	0.00	2207.47	15.74	0.04	3639.25	14.79	0.04	3641.44
500	25	30.00	0.09	420.61	48.87	0.00	2353.85	31.90	0.20	3658.67	31.32	0.20	3658.34
	50	24.58	0.14	521.95	41.37	0.00	3290.59	28.29	0.24	3658.95	27.52	0.24	3654.26
	75	20.98	0.08	652.78	40.47	0.00	3321.18	28.49	0.23	3664.69	27.66	0.23	3660.00
	100	12.68	0.05	751.30	53.14	0.00	1514.92	22.11	0.06	3667.39	21.21	0.06	3662.26
550	25	34.06	0.14	571.65	55.83	0.00	3605.38	36.75	0.19	3666.44	35.97	0.19	3666.87
	50	32.13	0.15	707.05	69.08	0.00	3278.58	39.19	0.25	3678.90	38.11	0.25	3674.31
	75	13.92	0.05	681.85	40.14	0.00	3226.88	20.21	0.24	3663.04	18.97	0.24	3661.60
	100	11.75	0.09	911.44	24.66	0.00	3401.85	20.54	0.43	3664.03	19.02	0.43	3660.42
600	25	25.56	0.14	683.79	41.33	0.00	3186.46	27.40	0.12	3698.79	26.85	0.11	3042.44
	50	30.42	0.08	1140.28	44.97	0.00	2871.13	34.15	0.03	3696.53	33.54	0.03	3691.63
	75	21.41	0.03	1075.67	54.57	0.00	3002.98	28.95	0.20	3723.01	28.33	0.20	3726.63
	100	10.43	0.07	1242.10	41.86	0.00	3643.51	18.03	0.07	3704.26	17.45	0.07	3702.21
Avg		20.09	0.19	320.68	38.39	0.01	1633.50	22.89	0.21	3190.62	20.58	0.20	2986.44

Table 1: Results for the set of standard QKP instances

Regarding the computational time in Table 1, it is clear that BDD has the least interesting performance while Caprara et al. (1999) is the fastest to converge. This is expected because the BDD B&B have no stopping criteria other than proving optimality, which is difficult to achieve, especially for large instances, while Caprara et al. (1999) has a simpler stopping criterion. Hence, most of the time, BDD stops only due to the time limit. However, we can observe an average difference of 3 minutes

between a BDD B&B with FC and with LEL. This means that the BDD B&B with FC solves more instances to optimality.

7.2.2 Dispersion and Densest Subgraph instances

This section presents the results on the dispersion and densest subgraph instances. The results are shown in Tables 2 and 3, wherein we report the average performance of 10 instances. On average, we observe that the BDD B&B approach reports the best results for these instances by far, with LEL being the best for the lower bound and FC for the upper bound.

However, for some categories of dispersion instances, the method of Fomeni et al. (2022) results in the best lower bounds (Table 2). For some of these upper bounds, the approach of Caprara et al. (1999) reports the best gaps for large instances. Otherwise, our BDDs have consistently good results, competitive with the best ones for these particular instances, and much better results on average.

instance	size	Caprara et al. (1999)			Fomeni et al. (2022)			BDD (LEL)			BDD (FC)		
		Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)
EXPO	25	21.81	0.02	3.01	0.19	0.19	0.63	0.00	0.00	0.15	0.00	0.00	0.10
	50	26.20	0.58	7.54	1.97	0.43	6.98	0.00	0.00	416.73	0.00	0.02	26.73
	100	29.44	0.93	21.33	62.11	0.39	184.86	19.17	0.00	2709.54	12.57	0.01	2317.66
	200	26.76	2.28	76.30	51.05	1.66	1225.56	24.74	0.00	3248.30	18.88	0.68	2900.12
	400	25.46	0.19	347.57	49.34	0.16	3601.45	29.68	0.00	3642.20	28.96	0.04	3315.18
GEO	25	14.95	0.13	2.82	0.14	0.14	4.68	0.00	0.00	0.09	0.00	0.00	0.03
	50	16.14	0.00	7.29	4.18	0.00	13.53	0.00	0.00	3.37	0.00	0.00	1.04
	100	16.47	0.00	22.77	40.63	0.00	42.44	4.88	0.00	2065.79	2.65	0.00	1635.46
	200	16.98	0.00	80.01	33.81	0.00	903.61	11.06	0.00	3603.94	9.03	0.00	3301.67
	400	16.91	0.01	356.55	53.74	0.00	3031.41	12.99	0.00	3637.32	12.06	0.00	3636.98
RAN	25	16.01	0.50	3.18	0.50	0.50	7.84	0.00	0.00	0.20	0.00	0.00	0.14
	50	20.98	0.49	4.66	4.19	0.41	8.37	0.00	0.00	906.58	0.00	0.00	28.42
	100	23.67	1.31	13.26	66.67	1.03	135.81	19.88	0.00	3246.48	14.53	0.01	2956.37
	200	20.18	0.34	78.12	49.66	0.29	2102.43	20.34	0.00	3604.65	18.03	0.06	2646.90
	400	24.81	1.02	339.18	69.48	0.82	3687.68	25.37	0.00	3639.74	23.89	0.14	3294.28
WGEO	25	9.84	0.81	2.52	0.00	0.00	0.26	0.00	0.00	0.08	0.00	0.00	0.04
	50	7.56	0.31	5.70	0.10	0.02	7.83	0.00	0.00	22.67	0.00	0.00	1.54
	100	8.25	0.18	19.62	0.07	0.00	3.10	15.34	0.27	3609.26	0.63	0.04	730.92
	200	7.94	0.11	89.53	30.18	0.00	38.17	18.67	0.23	3603.72	16.09	0.23	3605.10
	400	8.06	0.04	238.88	35.72	0.00	1067.41	21.03	0.11	3630.46	19.90	0.15	3629.75
KPEXPO	25	19.20	0.56	1.34	0.18	0.05	6.86	0.00	0.00	0.16	0.00	0.00	0.06
	50	18.88	0.35	3.51	0.25	0.00	9.01	0.00	0.00	15.35	0.00	0.00	1.83
	100	19.59	0.31	11.35	15.71	0.01	8.91	21.26	0.27	3606.75	7.54	0.24	3185.57
	200	19.39	0.19	48.96	39.57	0.00	151.33	24.84	0.35	3603.53	22.20	0.35	3605.01
	400	20.06	0.05	246.01	50.48	0.00	1098.06	26.50	0.08	3629.67	25.53	0.08	3628.54
KPGEO	25	6.40	0.14	2.29	0.03	0.00	6.83	0.00	0.00	0.05	0.00	0.00	0.03
	50	8.61	0.86	3.67	0.19	0.00	9.09	0.00	0.00	2.35	0.00	0.00	0.76
	100	8.10	0.49	11.92	4.94	0.00	13.57	13.69	0.28	3607.17	0.00	0.00	120.37
	200	7.65	0.09	50.82	44.58	0.00	38.72	18.47	0.30	3603.59	15.23	0.31	3605.35
	400	7.52	0.09	260.85	49.20	0.00	853.20	19.84	0.15	3629.12	18.81	0.15	3628.57
KPRAN	25	13.00	0.50	1.59	0.00	0.00	2.57	0.00	0.00	0.10	0.00	0.00	0.05
	50	12.72	0.14	3.57	0.19	0.00	9.21	0.00	0.00	3.11	0.00	0.00	0.90
	100	13.60	0.37	11.75	17.48	0.00	23.65	15.88	0.31	3608.41	0.00	0.00	715.44
	200	12.84	0.08	50.29	45.52	0.00	432.08	19.79	0.12	3604.11	16.47	0.12	3604.58
	400	12.65	0.06	251.00	44.76	0.00	1670.42	21.18	0.12	3630.51	20.18	0.12	3628.81
KPWGEO	25	4.25	0.36	3.05	0.07	0.00	3.37	0.00	0.00	0.13	0.00	0.00	0.04
	50	7.44	0.35	4.69	0.48	0.00	1.33	0.00	0.00	13.09	0.00	0.00	0.93
	100	7.30	0.18	11.46	4.68	0.00	10.05	17.71	0.16	3610.15	0.00	0.00	808.12
	200	6.87	0.12	48.76	32.81	0.00	26.04	19.22	0.16	3603.51	16.02	0.16	3604.83
	400	6.90	0.05	256.67	42.41	0.00	639.99	20.64	0.14	3629.15	19.49	0.17	3628.47
Avg		14.78	0.36	75.08	23.68	0.15	527.21	11.55	0.08	2124.78	8.47	0.08	1694.92

Table 2: Results for the dispersion instances

Concerning the densest subgraph instances reported in Table 3, our proposed approach computes the best gaps for most combinations of size and densities. The average deviation with the LEL cutset is 0.00%, indicating that it consistently obtains the BKS, and the average deviation for the FC is just 0.03%. Our method also reports the best gaps of 7.74% (LEL) and 6.78% (FC), against 10.58% of Caprara et al. (1999) and 27.11% of Fomeni et al. (2022).

instance	size	Caprara et al. (1999)			Fomeni et al. (2022)			BDD (LEL)			BDD (FC)		
		Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)
DSUB25	25	12.71	0.24	1.16	1.07	1.07	0.72	0.00	0.00	0.14	0.00	0.00	0.08
	50	10.70	0.10	2.89	1.15	0.22	2.14	0.00	0.00	79.07	0.00	0.00	1.38
	100	30.95	1.49	17.78	62.53	2.82	282.09	24.77	0.00	3607.41	17.19	0.07	2539.90
	200	17.98	1.56	30.14	57.50	1.71	2539.50	16.70	0.00	2532.77	16.45	0.08	2523.82
	400	30.68	1.02	133.25	63.48	1.16	3425.45	32.54	0.00	3637.33	32.06	0.20	3636.18
DSUB50	25	8.80	0.00	1.18	0.56	0.56	0.52	0.00	0.00	0.11	0.00	0.00	0.07
	50	2.64	0.10	3.01	0.86	0.86	1.74	0.79	0.00	395.45	0.00	0.00	29.33
	100	11.98	1.41	9.13	46.81	2.21	403.22	9.64	0.00	1865.99	7.37	0.02	2047.83
	200	13.58	0.36	31.32	39.29	0.78	1609.54	13.00	0.00	2163.80	12.58	0.05	1804.88
	400	12.43	0.24	137.95	56.26	0.39	3384.42	13.20	0.00	3635.30	12.74	0.03	3291.10
DSUB75	25	3.83	0.00	1.18	1.09	1.09	0.10	0.00	0.00	0.04	0.00	0.00	0.03
	50	0.84	0.00	2.96	0.34	0.34	2.64	0.00	0.00	10.93	0.00	0.00	0.28
	100	6.81	0.33	8.96	30.37	0.69	70.24	5.80	0.00	1804.21	2.18	0.00	1185.86
	200	9.98	0.61	33.88	55.02	0.92	2251.56	9.47	0.00	2523.97	9.16	0.02	2374.30
	400	9.04	0.27	141.55	55.70	0.50	3142.95	9.42	0.00	2914.28	9.16	0.01	2913.58
DSUB90	25	5.66	0.00	1.33	0.00	0.00	0.02	0.00	0.00	0.09	0.00	0.00	0.05
	50	4.07	0.03	3.14	0.16	0.15	0.96	0.61	0.00	417.45	0.00	0.00	21.44
	100	7.43	0.15	9.36	19.17	0.24	32.09	6.36	0.00	1803.89	4.88	0.00	1450.41
	200	4.14	0.09	32.66	18.86	0.13	1016.81	4.45	0.00	1803.78	3.75	0.01	1803.43
	400	7.40	0.05	145.20	32.04	0.06	3341.95	8.14	0.00	2556.05	7.99	0.01	2556.49
Avg		10.58	0.40	37.40	27.11	0.80	1075.43	7.74	0.00	1587.60	6.78	0.03	1409.02

Table 3: Results for densest subgraph instances

7.2.3 Hidden Clique instances

This section reports the results of the set of Hidden Clique instances in Table 4. Remarkably, our proposed approach reports the best gaps for these difficult instances, with the LEL cutset being the dominating one. Hence, we outperform state-of-the-art methods on both the lower and upper bound, improving the average gap and the deviation Δ by at least 10%. The method of Caprara et al. (1999) can only report good upper bounds but poor lower bounds; the method of Fomeni et al. (2022) reports the worst lower bounds and fails to report any significant upper bound within the time limit for instances of more than 100 items.

On the other hand, our proposed approach can prove optimality within the time limit of 1 hour for instances with up to 500 items. We observe the consequence of the quality of this result on the computational time. The average gap of our method (LEL) is of only 0.74% against 11.29% for Caprara et al. (1999) and 91.21% for Fomeni et al. (2022). The average deviation to the BKS of our method is of only 0.03%.

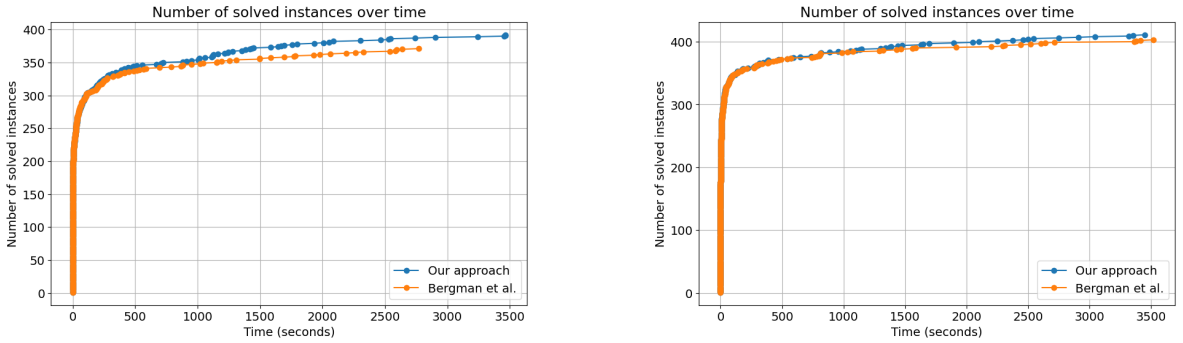
size	Caprara et al. (1999)			Fomeni et al. (2022)			BDD (LEL)			BDD (FC)		
	Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)	Gap (%)	Δ (%)	Time (s)
50	0.95	0.95	3.23	5.24	5.24	2.84	0.00	0.00	0.08	0.00	0.00	0.08
100	4.22	4.22	9.89	35.49	8.00	8.85	0.00	0.00	0.55	0.00	0.00	0.89
150	5.61	5.61	20.80	98.96	10.91	33.94	0.00	0.00	3.91	0.00	0.00	9.21
200	8.13	8.13	36.04	99.19	10.77	93.80	0.00	0.00	8.72	0.00	0.00	42.90
250	9.14	9.14	56.31	89.46	9.52	277.30	0.00	0.00	212.35	0.00	0.00	109.69
300	9.63	9.63	81.35	99.48	13.82	993.16	0.00	0.00	222.56	0.00	0.00	1066.87
350	10.07	10.07	114.39	99.57	13.20	1864.68	0.00	0.00	302.61	0.52	0.52	906.20
400	11.00	11.00	152.35	99.59	14.16	3482.54	0.00	0.00	37.70	0.00	0.00	576.86
450	12.19	11.94	201.75	99.65	14.61	3606.46	0.29	0.00	489.11	1.48	1.20	1268.15
500	12.47	12.01	235.55	99.68	13.93	3606.50	0.52	0.00	1791.60	2.77	2.28	2212.98
550	13.79	12.79	298.55	99.72	15.30	3610.72	1.34	0.20	1885.20	1.94	0.80	2541.42
600	12.83	11.67	404.84	99.75	16.07	3610.55	1.30	0.00	2606.07	3.33	2.08	2604.87
650	14.77	13.81	492.50	99.76	16.33	3619.66	1.10	0.00	1717.91	2.67	1.59	2637.26
700	15.05	14.10	583.21	99.78	17.53	3623.97	1.08	0.00	2167.54	3.17	2.12	3018.05
750	8.46	8.33	696.64	99.79	17.46	3624.21	0.14	0.00	1070.00	2.54	2.40	2686.69
800	16.35	15.63	816.96	99.81	17.98	3620.47	0.87	0.00	2091.89	2.72	1.87	2743.63
850	12.96	11.15	926.20	99.82	16.98	3617.30	2.02	0.00	3231.48	3.47	1.49	3494.68
900	15.06	13.75	1075.17	99.83	18.19	3612.50	1.68	0.14	2613.09	2.53	1.00	3168.53
950	15.68	13.87	1215.39	99.84	16.95	3615.42	2.11	0.00	2949.51	4.21	2.16	3261.11
1000	17.53	15.75	1564.63	99.85	18.11	3609.96	2.37	0.26	3745.17	3.55	1.47	3630.93
Avg	11.29	10.68	449.29	91.21	14.25	2506.74	0.74	0.03	1357.35	1.74	1.05	1799.05

Table 4: Results for the Hidden clique instances

7.3 BFS vs DFS

We now present and discuss a comparison between the DFS B&B proposed by Bergman et al. (2016) and our BFS B&B. Figure 3, reports the number of solved instances among all the sets of QKP instances used in the previous experiments within the same time limit of 1 hour. We run both B&B algorithms with a single thread, i.e., without parallelization, for a fair comparison with respect to the DFS, even if our BFS can be efficiently parallelized.

We observe in Figure 3 that our BFS outperforms the DFS by solving more instances. This is especially apparent when the LEL cutset is used, as shown in Figure 3a. Our approach solves 21 more instances than the DFS approach, with 19 instances from the set of Hidden Clique. The difference in the behavior is less flagrant when the FC cutset is considered, as one can see in Figure 3b.



(a) Last Exact Layer (LEL) **(b) Frontier Custset (FC)**

Figure 3: Cumulative number of solved instances over one hour of computation time.

While the conclusion from the computational experiments of Bergman et al. (2016) shows the superiority of LEL over FC, it is worth mentioning that our experiments show a slightly opposite behavior for all the QKP instances except the Hidden Clique ones.

8 Conclusion

In this paper, we have proposed a DP formulation for the QKP for BDDs. We enhanced this DP formulation by introducing long transitions, allowing for more efficient solution space encoding. Additionally, we proposed novel techniques for compiling approximate BDDs to compute tight bounds. Specifically, we introduced the concept of a local lower bound, enabling more effective node aggregation in both restricted and relaxed BDDs. We also introduced the use of MSTs in node clustering within relaxed BDDs, facilitating efficient computation of the dual bound and providing competitive results.

A particular result of this study is the introduction of the BFS B&B for BDDs. We present this approach as a technique that tightens both primal and dual bounds during the search. This has significant implications, as it enables the computation of not only a feasible solution to the QKP but also a tight dual bound, providing valuable insights into the solution's quality.

Our computational experiments have demonstrated the excellent performance of our proposed approach. We report results that, on average, improve the state-of-the-art duality gap of the QKP by up to 10% (Caprara et al., 1999; Fomeni et al., 2022). Our results are particularly noteworthy for the recently introduced set of instances by Schauer (2016), where our approach computes competitive lower and upper bounds, with an average gap of only 0.74% against 11.29% and 91.21% from the literature (Caprara et al., 1999; Fomeni et al., 2022).

A Appendix

Algorithm A.1 MST Algorithm for node clustering

procedure CLUSTER(G) ▷ G is the input graph
  $E \leftarrow$ sort edges of G by weight in non-decreasing order
  $\mathcal{T} \leftarrow$ empty set of connected components
  for $v \in V(G)$ **do**
    add $\{v\}$ to $\mathcal{T}$ ▷ Each vertex is initially its own component
  end for
  for $e \in E$ **do**
    $(u, v) \leftarrow e$
    $T_u \leftarrow$ component containing u in $\mathcal{T}$
    $T_v \leftarrow$ component containing v in $\mathcal{T}$
    if $T_u \neq T_v$ **then**
      merge T_u and T_v in $\mathcal{T}$
    end if
    if $|\mathcal{T}| = W$ **then**
      exit the loop
    end if
  end for
  return $\mathcal{T}$ ▷ Return set of connected components
end procedure

Algorithm A.2 Local upper bound calculation

procedure PROPAGATE(u, v) ▷ args: node u, integer v
  if $u_{lub} \geq v$ **then** ▷ if u has a higher upper bound, no improvements are required
    return
  end if
  $u_{lub} \leftarrow v$
  if $u \in \mathcal{S}$ **then** ▷ if u is in the exact cutset, no more computations are required
    return
  end if
  for $v \in U$ such that $a = (v, u) \in A$ **do**
    `propagate`(v, $u_{lub} + v(a)$) ▷ propagate the local upper bound to v, a parent node of u
  end for
end procedure

References

Alon, N., Arora, S., Manokaran, R., Moshkovitz, D., and Weinstein, O. (2011). Inapproximability of densest κ-subgraph from average case hardness. Technical report, School of Mathematical Sciences, Tel Aviv University.

Balas, E. (1975). Facets of the knapsack polytope. Mathematical Programming, 8:146–164.

Becker, B., Behle, M., Eisenbrand, F., and Wimmer, R. (2005). Bdds in a branch and cut framework. In International Workshop on Experimental and Efficient Algorithms, pages 452–463. Springer.

Bergman, D., Cire, A., Van Hoeve, W.-J., and Hooker, J. N. (2012). Variable ordering for the application of BDDs to the maximum independent set problem. In International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming, pages 34–49. Springer.

Bergman, D., Cire, A., Van Hoeve, W.-J., and Hooker, J. N. (2014a). Optimization bounds from binary decision diagrams. INFORMS Journal on Computing, 26(2):253–268.

Bergman, D., Cire, A., Van Hoeve, W.-J., and Hooker, J. N. (2016). Discrete optimization with decision diagrams. INFORMS Journal on Computing, 28(1):47–66.

Bergman, D. and Cire, A. A. (2017). On finding the optimal bdd relaxation. In Integration of AI and OR Techniques in Constraint Programming: 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings 14, pages 41–50. Springer.

Bergman, D., Cire, A. A., Sabharwal, A., Samulowitz, H., Saraswat, V., and van Hoeve, W.-J. (2014b). Parallel combinatorial optimization with decision diagrams. In Integration of AI and OR Techniques in Constraint Programming: 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings 11, pages 351–367. Springer.

Bergman, D., Cire, A. A., van Hoeve, W.-J., and Yunes, T. (2014c). BDD-based heuristics for binary optimization. Journal of Heuristics, 20:211–234.

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

Billionnet, A., Faye, A., and Soutif, É. (1999). A new upper bound for the 0–1 quadratic knapsack problem. European Journal of Operational Research, 112(3):664–672.

Cacchiani, V., Iori, M., Locatelli, A., and Martello, S. (2022). Knapsack problems — an overview of recent advances. Part II: Multiple, multidimensional, and quadratic knapsack problems. Computers & Operations Research, 143:105693.

Caprara, A., Pisinger, D., and Toth, P. (1999). Exact solution of the quadratic knapsack problem. INFORMS Journal on Computing, 11(2):125–137.

Castro, M., Cire, A., and Beck, J. (2022). Decision diagrams for discrete optimization: A survey of recent advances. INFORMS Journal on Computing, 34(4):2271–2295.

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

Chaillou, P., Hansen, P., and Mahieu, Y. (1989). Best network flow bounds for the quadratic knapsack problem. In Combinatorial Optimization: Lectures given at the 3rd Session of the Centro Internazionale Matematico Estivo (CIME) held at Como, Italy, 1986, pages 225–235. Springer.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). Minimum spanning trees. In Introduction to Algorithms, pages 624–642. MIT press.

Fennich, M. E., Fomeni, F. D., and Coelho, L. C. (2024). A novel dynamic programming heuristic for the quadratic knapsack problem. European Journal of Operational Research, 319(1):102–120.

Fomeni, F. D., Kaparis, K., and Letchford, A. (2022). A cut-and-branch algorithm for the quadratic knapsack problem. Discrete Optimization, 44:100579.

Fomeni, F. D. and Letchford, A. (2014). A dynamic programming heuristic for the quadratic knapsack problem. INFORMS Journal on Computing, 26(1):173–182.

Gallo, G., Hammer, P. L., and Simeone, B. (1980). Quadratic knapsack problems. In Mathematical Programming Studies, volume 12, pages 132–149. Springer.

Gillard, X., Coppé, V., Schaus, P., and Cire, A. A. (2021). Improving the filtering of branch-and-bound MDD solver. In International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pages 231–247. Springer.

González, J. E., Cire, A. A., Lodi, A., and Rousseau, L.-M. (2020). BDD-based optimization for the quadratic stable set problem. Discrete Optimization, page 100610.

Grygorash, O., Zhou, Y., and Jorgensen, Z. (2006). Minimum spanning tree based clustering algorithms. In 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06), pages 73–81. IEEE.

Johnson, E. L., Mehrotra, A., and Nemhauser, G. L. (1993). Min-cut clustering. Mathematical Programming, 62(1):133–151.

Létocart, L., Nagih, A., and Plateau, G. (2012). Reoptimization in Lagrangian methods for the 0-1 quadratic knapsack problem. Computers & Operations Research, 39(1):12–18.

Padberg, M. (1989). The boolean quadric polytope: some characteristics, facets and relatives. Mathematical Programming, 45:139–172.

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

Schauer, J. (2016). Asymptotic behavior of the quadratic knapsack problem. European Journal of Operational Research, 255(2):357–363.

Sherali, H. D. and Adams, W. P. (1990). A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. SIAM Journal on Discrete Mathematics, 3(3):411–430.

Tjandraatmadja, C. and Van Hoeve, W.-J. (2019). Target cuts from relaxed decision diagrams. INFORMS Journal on Computing, 31(2):285–301.

Wolsey, L. A. (1975). Faces for a linear inequality in 0–1 variables. Mathematical Programming, 8(1):165–178.