Annals of Operations Research
https://doi.org/10.1007/s10479-024-06025-1

ORIGINAL RESEARCH

Optimal solving of a binary knapsack problem on a D-Wave quantum machine and its implementation in production systems

Wojciech Bożejko[1] · Anna Burduk[2] · Jarosław Pempera[1] · Mariusz Uchroński[3] · Mieczysław Wodecki[4]

Received: 4 October 2023 / Accepted: 16 April 2024

Abstract
The efficient management of complex production systems is a challenge in today's logistics. In the field of intelligent and sustainable logistics, the optimization of production batches, especially in the context of a rapidly changing product range, requires fast and precise computational solutions. This paper explores the potential of quantum computers for solving these problems. Traditional computational methods are often limited when it comes to optimizing complex logistics systems. In response to these challenges, the paper proposes the use of a hybrid algorithm that combines quantum technologies with classical computational methods. Such integration allows the computational power of both types of technologies to be harnessed, leading to faster and more efficient identification of optimal solutions. In this work, we consider the knapsack problem, a classic NP-hard optimization problem that is commonly used to verify the effectiveness of new algorithm construction methods. The algorithm presented is based on the Branch and Bound method and aims to ensure solution optimality in the context of the non-determinism of quantum computers. Within the algorithm, computations are performed alternately on a classical processor and a quantum

✉ Wojciech Bożejko
wojciech.bozejko@pwr.edu.pl

Anna Burduk
anna.burduk@pwr.edu.pl

Jarosław Pempera
jaroslaw.pempera@pwr.edu.pl

Mariusz Uchroński
mariusz.uchronski@pwr.edu.pl

Mieczysław Wodecki
mieczyslaw.wodecki@pwr.edu.pl

1 Department of Control Systems and Mechatronics Wrocław, University of Science and Technology, Janiszewskiego 11/17, 50-372 Wrocław, Poland

2 Faculty of Mechanical Engineering, Wrocław University of Science and Technology, Ignacego Łukasiewicza 5, 50-371 Wrocław, Poland

3 Wroclaw Centre for Networking and Supercomputing, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland

4 Department of Telecommunications and Teleinformatics, Wrocław University of Science and Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland

Published online: 13 May 2024

processor. In addition, the lower and upper bounds of the objective function are computed in constant time using the D-Wave quantum machine.

Keywords Discrete optimization · Knapsack problem · Quantum annealing

1 Introduction

Packing problems are very important in the practice of managing production processes, their automation and logistics. They enable the modeling of practical issues related to transport, managing warehouse space, loading goods, designing systems with a high scale of integration or managing computer networks and databases. The construction of effective methods for determining solutions allows for the reduction of transport and storage costs.

In the paper, we consider the binary knapsack problem (PP), i.e., the one-dimensional packing problem. In this problem, the data is a collection of things (elements) each of which has a fixed weight and value. You need to designate a subset of items with a maximum sum of values whose total weight does not exceed the capacity of the knapsackk. In the literature, there are many such issues related to many additional assumptions and constraints. In particular, one- and multi-criteria problems of two- and three-dimensional packaging of elements, including irregularly shaped ones.

In this paper, we present an optimal hybrid algorithm for solving the knapsack problem, the design of which is based on the Branch and Bound method (B&B). Although the quantum annealing algorithm does not guarantee the optimality of the determined solution (see Pusey-Nazzaro & Date, 2020 for the knapsack problem), it can be used in the construction of an exact algorithm, as it improves its efficiency through non-deterministic control of the solution space search process. We propose a new approach, consisting in determining the upper and lower bounds of the objective function on the quantum annealer. For the determination of the upper bound, we formulate the problem of binary quadratic programming with constraints, translated natively onto QUBO, which is a natural way of formulating computational tasks for the D-Wave machine. In contrast, for the determination of the lower bound, we apply the Lagrange relaxation.

1.1 State of the art

The Knapsack problem, despite the simplicity of the formulation (linear objective function and one linear constraint), belongs to the class of the most difficult, NP-hard, discrete optimization problems. Already, for 60 items, the number of solutions is 2^{60} solutions. If we checked a billion solutions in 1 s, it would take over 30 years to check all combinations. This problem is important both from the point of view of practice and theory. Biglar (2018) lists over 20 practical issues directly related to the packaging problem considered in this work, including production planning, power allocation management, resource management as well as power allocation planning. Many interesting practical examples of the Knapsack Problemm applications and methods of solving them are also presented in the works of Laabadi et al. (2018), Wilbaut et al. (2008), and Cho (2019). The methods and constructions of algorithms presented in this paper can be adapted to model and solve more complex industrial and logistic processes.

Methods for solving the Knapsack Problem have a long and rich history. The first optimal algorithms based on dynamic programming and division and constraints were published at

the turn of the 1950 s and 1960 s. They allowed solving examples of up to a dozen items. Too small from the point of view of practical applications. Despite the passage of many years, exact methods, including pseudo-polynomial algorithms, i.e. polynomial with respect to the knapsack payload, are successively developed and published. An overview of the methods and algorithms are presented, among others, in the following works: Coniglio et al. (2021), Zavala-Diaz et al. (2019); Rizk-Allah and Hassanien (2018) and Shen et al. (2019). Many construction algorithms have also been published, mainly based on the greedy method. They are easy to implement, work quickly, but usually, the designated solutions are far from optimal. They are used primarily to determine starting solutions in optimal algorithms.

A significant breakthrough occurred in the 1990 s. In response to the expectations of practitioners, there has been an intensive development of approximate methods, including the now classic metaheuristics: Tabu Search, Simulate Annealing, and evolutionary algorithms such as genetic algorithm, particle swarm optimization, differential evolution, ant ant colony optimization. The work by Zhang (2011) presents an algorithm using artificial intelligence; Refaei et al. (2020) proposes neural networks and machine learning. Approximation algorithms and extensive analyses of the performed computational experiments are included in the works: He et al. (2024), Refaei et al. (2020), and Zhang (2011). An overview of packaging problems and methods of solving them is presented in the monograph of Kellerer et al. (2004). An extensive review of the current literature is provided in Cacchiani et al. (2022), Wang et al. (2022), and Barakat et al. (2016).

The possibility of multi-processor computing has resulted in a significant shift in the size of instances that can be solved exactly in an acceptable time. In addition to the exact and approximation algorithms that have been known for years (Zavala-Diaz et al., 2019; Vu & Derbel, 2016; Vasilchikov, 2018), parallel versions of algorithms inspired by nature have also been published (He et al., 2024), as well as: neural networks, artificial intelligence (Zhang, 2011; Ji et al., 2017) and learning techniques (Refaei et al., 2020).

Great hopes are attached to the possibility of performing calculations on quantum computers. Currently, they have little practical application in solving (exactly) discrete optimization problems. The main limitation is the small number of qubits. In recent years, they have been developing intensively. It can be expected that in the coming years practical examples of very large size will be solved within a reasonable time. Among quantum approaches, Pusey-Nazzaro and Date (2020) claims that adiabatic quantum optimization is unable to provide the optimal solution to a variety of small knapsack problems. In the work of Li et al. (2023), the authors use a new approach to solving the Knapsack Problem by using quantum-behaved particle swarm optimization. Approximation algorithms are presented in Dam et al. (2021). Bożejko et al. (2022) proposed a distributed quantum annealing algorithm for a single-machine scheduling problem with total weighted tardiness criterion. In terms of optimal solving of optimization problems (Bożejko et al., 2024, 2023) proposed exact algorithms for solving single-machine scheduling problems on a quantum machine in the CPU-QPU hybrid approach. In the work of Yarkoni et al. (2022) is presented a review of the literature on the use of a quantum annealer, which can be used in combinatorial optimization.

The effective use of modern quantum computing technologies, including D-Wave quantum machines, is associated with a number of specific limitations. It requires, among others, formulating the problem to be solved in the form of a binary quadratic programming task (Quadratic, Unconstrained Binary Optimization, QUBO). This is often very difficult, especially when the objective function is minimax.

Calculations on D-Wave quantum computers for a wider range of users have been possible practically since 2011 (D-Wave One machine). The size of currently possible solvable examples (number of elements) is limited by the number of qubits (today, over 5000 for

D-Wave Pegasus technology). The systematic development of hardware suggests that in the near (near) future, it will be possible to solve examples of much larger sizes than on classic computers.

The quantum computers currently in use represent two concepts of quantum computing. The first is quantum computers based on the gate model, with quantum gates operating on qubits and programs written as a circuit (*circuit*). Quantum gates (Pauli's, Hadamard, Toffoli, CNOT, etc.), unlike classical logic gates in electronic computing, are reversible. This type of computer is currently being developed by IBM and Google.

The second type of quantum machines are the so-called *quantum annealers* realizing adiabatic quantum computing. They use a physical process to obtain the state of the qubits of a quantum system representing the discrete optimization problem being solved in a minimal-energy state. The advantage of quantum annealers is that the number of qubits can be much larger than that currently available in gate-based quantum computing systems (noise is not as significant as in quantum gate models); however, their use is limited only to specific cases related to the representation of the problem to be minimized. This type of device is being developed by NEC and D-Wave.

1.2 Quantum annealing

It is a promising computational method using quantum phenomena to solve challenging optimization tasks. A quantum machine implementing quantum annealing handles a certain limited class of optimization problems, which can be formulated as problems of Quadratic Unconstrained Binary optimization (QUBO). There is currently an intensive search for applications of this technology in practice, learning about its capabilities and limitations – the largest currently available quantum machines of this type have 5640 qubits. The quantum annealer, through the continuous evolution of the quantum system, searches for the minimum energy of the Ising Hamiltonian (see Ajagekar et al., 2020; Denkena et al., 2021).

In practice, the tasks formulated for a quantum machine implementing quantum annealing are in the form of an Ising or QUBO model, with the translation of problems between these models being natural. The Ising model is used in statistical mechanics with a criterion function:

$$E_{Ising}(s) = \sum_{i=1}^{N} h_i s_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} J_{i,j} s_i s_j, \tag{1}$$

where s_i, $i = 1, 2, \ldots, N$ express spins of $+1$ and -1, while the linear coefficients corresponding to the cubit deviations are h_i and the quadratic coefficients corresponding to the coupling forces are $J_{i,j}$. In the QUBO model, on the other hand, the function to be minimized is of the form

$$f(x) = \sum_{i=1}^{N} Q_{i,i} x_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} Q_{i,j} x_i x_j, \tag{2}$$

where Q is an upper-diagonal matrix with $N \times N$ real weights, x is a vector of binary variables, $x_i \in \{0, 1\}$, $i = 1, 2, \ldots, N$.

QUBO is an unconstrained model, meaning that in practice, any constraints on the problem must be included in the objective function. Some of the computation models (solvers) available within the D-Wave Ocean libraries are constrained models, e.g. *LeapHybridCQMSampler*—Constrained Quadratic Model (CQM), for which the translation of a constrained problem to an unconstrained problem takes place inside the solver. The problem formulation

for the CQM model takes the form of a minimization:

$$\sum_{i=1}^{N} a_i x_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} b_{i,j} x_i x_j + c, \tag{3}$$

with constraints:

$$\sum_{i=1}^{N} a_i^{(m)} x_i + \sum_{i=1}^{N} \sum_{j=i+1}^{N} b_{i,j}^{(m)} x_i x_j + c^{(m)} \propto 0, \quad m = 1, 2, \ldots, M \tag{4}$$

where x_i, $i = 1, 2, \ldots, N$ are binary or integer variables, $a_i, b_{i,j}, c, i, j = 1, 2, \ldots, N$, are real values, relation $\propto \in \{\geq, \leq, =\}$ and M is the number of all constraints.

2 Discrete knapsack problem

In this paper, we consider a well-known problem in the literature called the Discrete Knapsack Problem (in short DK), often found in production planning issues (Camargo et al., 2012). There is a set of *items* $\mathcal{N} = \{1, 2, \ldots, n\}$ and capacity of the knapsack W. With every item $i \in \mathcal{N}$ the following notions are linked:

w_i weight of the item i,
v_i value of the item i.

One has to determine the subset of items with a maximum sum of values whose sum of weights does not exceed the load capacity of the knapsack W. Each solution can be represented by a binary sequence

$$\mathbf{x} = (x_1, x_2, \ldots, x_n), \tag{5}$$

where $x_i \in \{0, 1\}$, $i \in \mathcal{N}$. If $x_i = 1$ then that means that i-th item is 'packed' in the knapsack.

As a discrete linear programming problem, the knapsack problem is formulated as follows:

$$\text{maximize}: \ F(\mathbf{x}) = \sum_{i=1}^{n} v_i x_i \tag{6}$$

subject to:

$$\sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0, 1\}, \ i \in \mathcal{N}. \tag{7}$$

Despite the simplicity of its formulation (a linear criterion and a single linear constraint), it belongs to the class of *NP*-hard problems. These are problems for which no algorithms are currently known for solving them with polynomial computational complexity.

By $\mathcal{X}$ we denote the set of *feasible solutions*, i.e., n-element binary sequences,

$$\mathcal{X} = \{(x_1, x_2, \ldots, x_n) : \sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0, 1\}, i \in \mathcal{N}\}. \tag{8}$$

In view of this, the knapsack problem reduces to determining the element in $\mathcal{X}$, that maximizes the criterion (6), i.e., the optimal solution $\mathbf{x}^* \in \mathcal{X}$ such that

$$F(\mathbf{x}^*) = \max\{F(\mathbf{x}) : \ \mathbf{x} \in \mathcal{X}\}. \tag{9}$$

In the further part of this paper, we present a method and algorithm for optimal solving the knapsack problem and procedures for counting, on a D-Wave quantum machine, the lower and upper bounds.

3 Solution method

If the solutions to the DK problem are represented by binary sequences then, according to (9), solving the problem reduces to determining the optimal element in the set of feasible solutions $\mathcal{X}$. The process of viewing the elements of this set will be represented by a directed *solution tree* $\mathcal{H}$.

The root of the tree (the only vertex at level zero) is any (starting) sequence $\mathbf{x} \in \mathcal{X}$, in which the set of free elements $\mathcal{K} = \mathcal{N}$. In the root, at most two vertices at level one, are generated from each free element. These are created by determining the value of one undetermined element z, i.e., by substituting $x_z = 0$ or $x_z = 1$. Similarly, from each vertex of level one, two more vertices can be generated by determining the value of one of the $n - 1$ free elements. These form the 2nd level of the tree. Proceeding similarly, we can generate vertices (solutions) at subsequent levels. Since the tree represents the set of $\mathcal{X}$, feasible solutions, we fix the z-th free element, i.e., we adopt $x_z = 1$, only if the solution generated in this way satisfies the constraint (8). In the algorithm for searching the solution tree $\mathcal{H}$ based on the B&B method, the following procedures play an essential role: generating a new vertex, backtracking to a lower level of the tree, and computing the lower and upper bounds on the value of the criterion function. The constraints of the function allow certain subtrees to be omitted from the search process of the $\mathcal{H}$ tree without losing the optimality of the solution. By $\mathcal{H}(\mathbf{x})$we denote a subtree in $\mathcal{H}$, whose root is a vertex of the $\mathbf{x}$.

Generating a new vertex. Let $\mathbf{x} \in \mathcal{X}$ be a vertex of $\mathcal{H}$ on the h-th level. We generate a new vertex $\mathbf{y}$ (on (h+1)-th level) by fixing a job z from the set of candidates $\mathcal{K}$. Then we add an arc ($\mathbf{x}$,$\mathbf{y}$) to the tree $\mathcal{H}$. In each successor of the vertex $\mathbf{y}$ the job z is fixed. The order in which candidates are selected for determination has a significant impact on the possible rejection of a subtree in the process of searching the solution tree.

Backtracking. Let us assume that the vertex $\mathbf{y}$ was generated from $\mathbf{x}$ by determining the element z. Therefore, if we backtrack from $\mathbf{y}$ to vertex $\mathbf{x}$ then from the set of candidates in vertex $\mathbf{x}$ we remove the element z. The backtracking operation is performed if the set of candidates at the vertex is an empty set. If it is the root of the tree, the algorithm terminates. Solving the DK problem thus boils down to determining the vertex (sequence) in the tree $\mathcal{H}$ of the maximal value of the objective function (6).

Lower and upper bounds. We consider the vertex of the $\mathbf{x}$ of the solution tree $\mathcal{H}$. The lower bound on the value of the criterion function in the subtree $\mathcal{H}(\mathbf{x})$ is defined as

$$LB(\mathbf{x}) = \sum_{i=1}^{n} v_i x_i + LBQ(\mathbf{x}), \tag{10}$$

where the first component is the sum of the values of the items determined in $\mathbf{x}$ (the items already selected for the knapsack). On the other hand, $LBQ(\mathbf{x})$ is a lower bound on the of the sum of the values of the items with which we can still complete the knapsack. However, their weight must not exceed $W - \sum_{i=1}^{n} w_i x_i$. It is easy to see that $LB(\mathbf{x})$ is the value of some solution $\mathbf{x}^*$ of the subtree $\mathcal{H}(\mathbf{x})$. Element $x_i^* = 1$, $i = 1, 2, \ldots, n$, if $x_i = 1$ or i-th element was chosen when computing the value of $LBQ(\mathbf{x})$. Then, the value of the criterion function $F(\mathbf{x}^*) = LB(\mathbf{x})$.

In turn, the upper bound of the value of the criterion function

$$UB(\mathbf{x}) = \sum_{i=1}^{n} v_i x_i + UBQ(\mathbf{x}), \tag{11}$$

where the first component is the sum of the values of the items fixed in $\mathbf{x}$ (items already selected to be in the knapsack), whereas $UBQ(\mathbf{x})$ is the upper bound of the sum of the values of items that are not yet determined (to the knapsack).

The values of the lower and upper bounds have an important impact on the B&B algorithm workflow. This is because if $LB(\mathbf{x}) \geq UB(\mathbf{x})$, then without losing the optimality of the solution, the subtree $\mathcal{H}(\mathbf{x})$, including the vertex of $\mathbf{x}$ can be omitted. In this way, the computation time can be significantly reduced. In order to obtain the largest possible value of the lower bound and the smallest possible value of the upper bound, we will determine both on the D-Wave quantum machine.

The most important elements of the proposed B&B method are methods of determining the upper estimate of the optimal value of the objective function and determination of the best possible solution for each node of the tree. Let $UB(\mathcal{N}, W)$ be the upper estimate of the value of the objective function for the set of elements $\mathcal{N}$ for a knapsack of the load W, then the value of the upper estimate of the optimal value of the objective function for the node described by the quadratic $(h, \alpha, x, \mathcal{K})$ can be determined from the formula

$$UB(h, \alpha, x, \mathcal{K}) = \sum_{i=1}^{h} v_{\alpha(i)} x_{\alpha(i)} + UB(\mathcal{K}, W - \sum_{i=1}^{h} w_{\alpha(i)} x_{\alpha(i)}). \tag{12}$$

Further, let $BF(\mathcal{N}, W)$ be the value of the objective function determined for a set of elements and a knapsack of W, then value of the objective function for the best solution representing the node described by the tuble $(h, \alpha, x, \mathcal{K})$ is

$$BF(h, \alpha, x, \mathcal{K}) = \sum_{i=1}^{h} v_{\alpha(i)} x_{\alpha(i)} + BF(\mathcal{K}, W - \sum_{i=1}^{h} w_{\alpha(i)} x_{\alpha(i)}). \tag{13}$$

If $UB(h, \alpha, x, \mathcal{K}) \leq F(x^*)$, where x^* is the best solution found so far then it is removed from the tree (cut in the B&B method). The values of $UB(\mathcal{N}, W)$ and $BF(\mathcal{N}, W)$ will be determined on the D-Wave quantum computer. This requires their transformation to QUBO form.

3.1 Determination of constraints on a D-Wave quantum machine

Tasks formulated for a solution by a D-Wave quantum machine must be in the form of discrete quadratic programming with constraints and minimization of the objective function (QUBO). We will therefore reduce the tasks of counting the value of the lower and upper bounds to this form. To simplify the notation, let us assume that the free elements in the solution $\mathbf{x}$, are numbered consecutively with numbers from 1 to t.

Lower bound. To determine the value of $LBQ(\mathbf{x})$ in the formula for the lower bound (10), we will formulate a certain subproblem (abbreviated as **PDK**) of the initial knapsack problem. Let $P = W - \sum_{i=1}^{n} w_i x_i$ be the load capacity of the knapsack. One should choose such a subset of the set of elements with indices $i = 1, 2, \ldots, t$ that the sum of the value of elements is maximal and the sum of the weights does not exceed the load capacity P. Thus,

one should

$$\text{maximize}: f(\mathbf{y}) = \sum_{i=1}^{t} v_i y_i \tag{14}$$

with constraints:

$$\sum_{i=1}^{t} w_i y_i \leq P, \quad y_i \in \{0, 1\}, i = 1, 2, \ldots, t. \tag{15}$$

Since $\max f(x) = -\min(-f(x))$ thus the optimization problem (14)–(15) represents the binary optimization problem with constraints, which can be natively translated on a D-Wave to a QUBO model. If $\mathbf{y}^*$ is its solution, then we assume $LBQ(\mathbf{x}) = f(\mathbf{y}^*)$.

Upper bound. We use the Lagrange relaxation method to determine the value of the upper bound $LBQ(\mathbf{x})$, for the PDK problem defined in (14)–(15), The Lagrange function with multiplier u, for the variables $y = (y_1, y_2, \ldots, y_t)$, takes the form:

$$\begin{aligned} L(y, u) &= \sum_{i=1}^{t} v_i y_i + u(w_1 y_1 + w_2 y_2 + \ldots + w_t y_t - W) \\ &= \sum_{i=1}^{t} L_i(y_i, u) - uW, \end{aligned} \tag{16}$$

where

$$L_i(y_i, u) = y_i \left(v_i + w_i u\right).$$

Maximalization $L(x, u)$ with respect to individual variables x_i, for fixed values of u, can be performed independently.

Let us note that for *any* $u \leq 0$, and the optimal solution y^* of the PDK problem, the following lemma holds:

Lemma 1 *(Bożejko et al., 2024) For any value $u_i \leq 0$, $i = 1, 2, \ldots, t$ of the Lagrange multipliers, the value $L(y, u)$ of the Lagrangian function is an upper bound on the optimal objective function value $\sum_{i=1}^{t} v_i y_i^*$ of the PDK problem.*

Using the above lemma, it can be shown that

$$UBQ(\mathbf{x}) = \min_{u \in U} \left\{ \sum_{i=1}^{t} \max_{y_i \in \{0,1\}} L_i(y_i, u) - uW \right\} \tag{17}$$

is an upper bound on the optimal value of the maximized objective function $\sum_{i=1}^{t} w_i y_i^*$ for the PDK problem. Thus, to compute $UBQ(\mathbf{x})$, the upper bound of the Lagrange function for the PDK problem on the D-Wave computer, we determine the values of the vector u using quantum annealing by solving the following QUBO task:

$$\min_{u \in U, y \in A} \left\{ \sum_{i=1}^{t} L_i(y_i, u) - uW \right\}, \tag{18}$$

where A is the set of solutions admissible for the PDK problem, given the constraints:

$$L_i(y_i, u) \geq L_i(0, u), \tag{19}$$

and

$$L_i(y_i, u) \geq L_i(1, u), \tag{20}$$

for $u \leq 0$.

Table 1 Problem instance data

i	1	2	3	4	5	6	7	8	9	10	11	12
w_i	94	416	992	649	815	422	791	667	598	7	766	893
p_i	96	417	993	651	816	423	792	669	600	9	768	894

In total, the total number of constraints is $2t$. The variables y_i take two values 0 or 1. In view of this, the maximization over the vector y in the expression (17) is replaced by the inequalities (19) and (20)—paradoxically, the maximization over y is obtained in the formula (18) even though formally a minimum is written, but de facto it only means a run over all values of the variable y. This formulation is necessary in order to run the task on a D-Wave machine that only performs minimization with respect to the given variables.

4 Exact hybrid algorithm

By performing calculations on a quantum machine, it is possible to determine the lower and upper bounds of the criterion function at a vertex of the solution tree in a short time (in our experiments the QPU time was 15–32 ms). In the hybrid B&B algorithm, the lower and upper bounds are determined on the QPU of the D-Wave quantum machine, and the remaining computations are performed on the CPU. When searching the solution tree of $\mathcal{H}$, a *best-first* strategy is used with a priority queue implemented in the heap structure (variable *Heap*) containing the values of the upper bounds. Tuples are remembered at the vertices of the mound: (*heap*, x,α *candidate set, upper bound, value of objective function*), where x is the vertex of the tree and α the set element generating the successor of x in the tree.

The algorithm starts from step 1, fetching the tuplet from the priority queue (heap). In step 2, a decision is made whether to develop a new subtree. In step 3 of the algorithm, for each candidate, $k \in \mathcal{K}$, the feasibility of the solution is checked (step 3.2), and then the estimates are calculated. If the generated solution is not feasible or the value of the upper estimate is not greater than the lower estimate (the best one determined so far), the vertex corresponding to the value $x_k = 1$ is not generated. The element k is removed from the set of candidates (step 3.3). The selection of the best candidate to determine and the updating of the best solution is performed in step 4. In steps 5 to 8 the new vertices of the solution tree are generated. In step 8.1, selected candidates are put to the $Heap$ (see Fig. 1).

4.1 Case study

Using a small example of a knapsack problem, we show the progression of the classical B&B algorithm (running on the CPU) and the hybrid `B&B_CPU_QPU` algorithm run on the QPU and CPU. The set of items $\mathcal{N} = \{1, 2, \ldots, 12\}$. The weights and the values of the items are provided in Table 1. knapsack capacities $W = 994$.

The solution tree of the B&B algorithm is shown in Fig. 2 for a test instance data from Table 1. Each vertex in the tree contains: the iteration number (top left corner), the value of the upper bound (top right corner), the items in the knapsack (center), and the sum of the values of the items in the knapsack (bottom right corner). The absence of descendant vertices means that they have been cut off. There were 20 vertices generated during the calculation. The optimal solution was obtained in 3 iterations. There were still 18 iterations until the

Hybrid `B&B_CPU_QPU` **Algorithm:**

Output x^* – optimal solution;

Heap : priority queue of solutions sorted in non-decreasing order by their value of criterion function;

$x^* = \arg\max_x BF(0, \alpha, x, \mathcal{N})$;
Put(*Heap*,$(0, x^*, \alpha, \mathcal{N}, UB(0, \alpha, x, \mathcal{N}), F(x^*))$;
while *Heap*$\neq \emptyset$ **do**
1 Get(*Heap*,(h,x,α, $\mathcal{K}$, UB));
2 **if** $(\lfloor UB \rfloor \leq F(x^*))$ **then continue**;
3 **for** each $k \in \mathcal{K}$ **do**
3.1 $\alpha(h+1) = k$; $x_k = 1$;
3.2 **if** $\sum_{i=1}^{h+1} x_{\alpha(i)} w_{\alpha(i)} \leq W$ **and**
$UB(h+1, \alpha, x, \mathcal{K} \setminus k) > F(x^*)$ **then**
3.2.1 $y^k = \arg\max_x BF(h+1, \alpha, x, \mathcal{K} \setminus k)$;
3.3 **else** $\mathcal{K} = \mathcal{K} \setminus k$;
4 $z = \arg\min_{k \in \mathcal{K}} F(y^k)$; **if** $F(y^z) < F(x^*)$ **then** $x^* = y^z$;
5 $x_z = 0$;
6 **if** $UB(h+1, \alpha, x, \mathcal{K} \setminus z) > F(x^*)$ **then**
7.1 Put(*Heap*,($h+1$,x,α,$\mathcal{K} \setminus z$,
$UB(h+1, \alpha, x, \mathcal{K} \setminus z)$,$F(x)$));
7 $x_z = 1$;
8 **if** $UB(h+1, \alpha, x, \mathcal{K} \setminus z) > F(x^*)$ **then**
8.1 Put(*Heap*,($h+1$,x,α,$\mathcal{K} \setminus z$,
$UB(h+1, \alpha, x, \mathcal{K} \setminus z)$,$F(x)$));
end while

Fig. 1 Hybrid quantum algorithm

calculation was fully completed. Fig. 3 shows a fragment of the solution tree illustrating the flow of calculations between the CPU and QPU to generate the vertexes.

The hybrid algorithm `B&B_CPU_QPU`, at the root of the solution tree, determined a solution $x_1 = 1, x_{10} = 1$ and $x_{12} = 1$ (the remaining elements are 0) with a value of 999. At this vertex, the upper estimate of the optimal value of the objective function is $UB = 1001.9$. Next, vertices from level 1 of the tree were generated sequentially. Some results are included in Table 2. Row one (denoted by z) shows the variables that were sequentially given the value 1. Row two shows the values of the upper bounds, and row three shows the values of the best solution found.

For the vertex of the tree generated by determining $x_3 = 1$, the value of the upper bound $UBQ = 993$ is less than $F(x^*) = 999$, thus item 3 is removed from the set of candidates. The same is true for items 2, 4, 5, 6, 7, 8, 9, 11, and 12. Finally, the set of candidates to be determined $\mathcal{K} = \{1, 10, 12\}$.

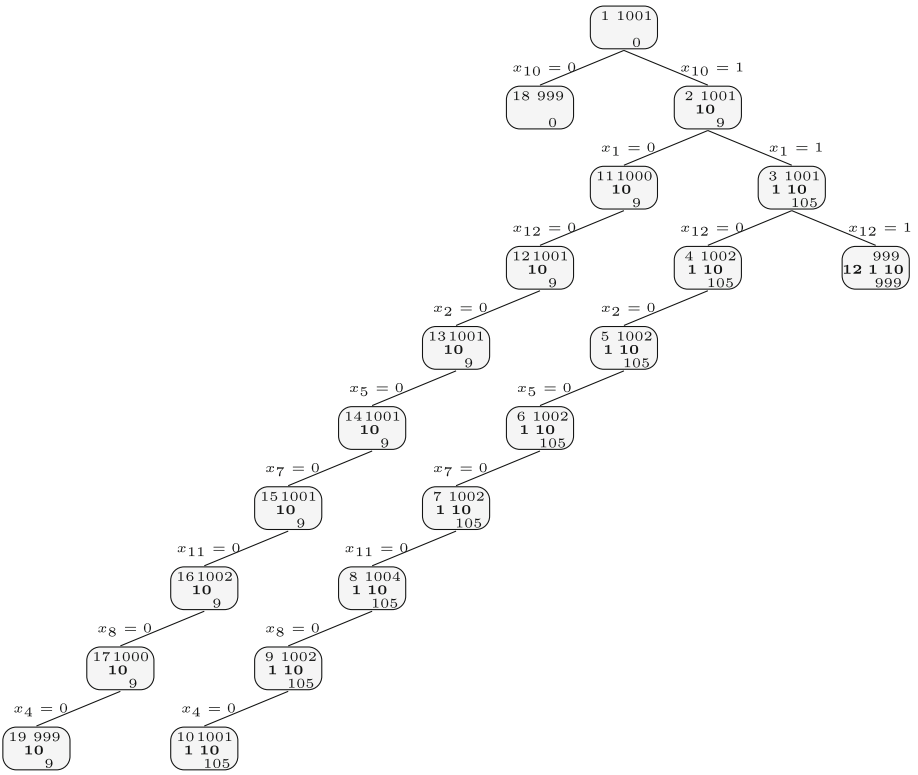

Fig. 2 The solution tree of the B&B algorithm

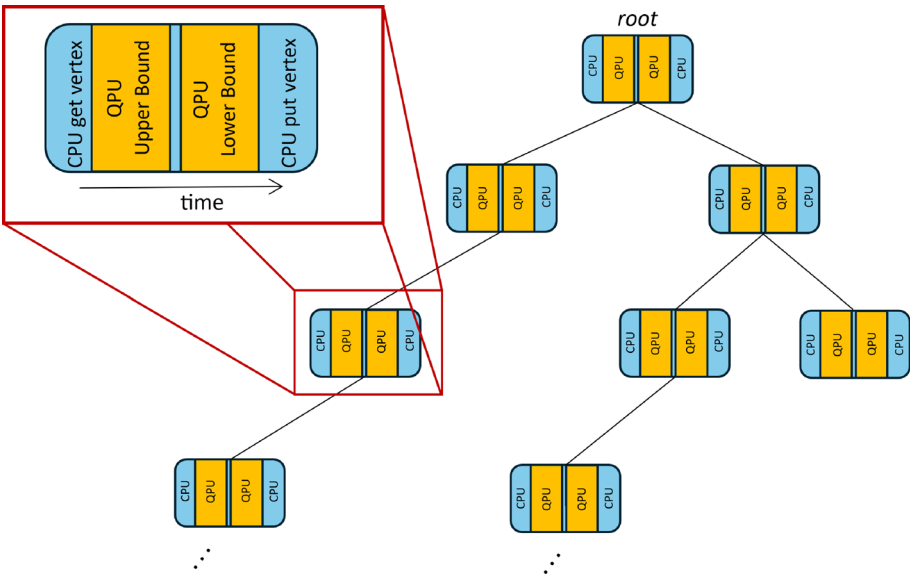

Fig. 3 Flow of computation by the CPU/QPU

Table 2 Values determined for the *root* in the first iteration of the algorithm

z	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
UBQ	4260	944	993	756	921	528	897	774	705	11, 688	873	999
$F(x^*)$	999	–	–	–	–	–	–	–	–	901	–	–

Let $z = 1$ be the candidate selected in Step 5. For $x_1 = 0$ the value of UB is 903, while for $x_1 = 1$ it is 999. Both values are smaller than $F(\pi^*)$, therefore no descendant nodes will be generated, which terminates the algorithm.

5 Computational results

This section aims to experimentally investigate the efficiency of quantum computing on a medium to large data sizes.

We evaluate the performance of quantum computing on state-of-the-art instances of the knapsack problem taken from OR-Library (they are available here Pempera & Uchroński, 2023). We especially focus on 30 hard instances ranging from $n = 100$ to $n = 500$ items. The original test instances have more than one constraint; therefore, we only considered the first constraints in the problem under consideration.

The calculations on the quantum machine were limited to calculating only UB and LB for the root of the B&B search tree for each instance. This is related to a long time of initial data processing in the cloud by the provider of the quantum computing service and/or its business policy. This time is not less than 5 s for one run of calculations. In addition, calculations were carried out on a classic computer using the GUROBI package. As a result of these calculations, optimal values were obtained.

5.1 D-Wave quantum computing environment

Quantum calculations were performed using *LeapHybridCQMSampler* solver, which implements quantum annealing in hardware. CQM solver runs with an Advantage backend for the quantum portion of the quantum-classical hybrid solver. Advantage QPUs are based on the Pegasus graph topology with size P16 containing at least 5000 qubits with 15 couplers per qubit, totaling at least 35.000 couplers. Classical calculations were performed on a Dell OptiPlex 7040 computer with an Intel Core i5-6500 processor and 12 GB RAM.

The Table 3 presents the results of experimental studies. Column 4 shows the total access time to the quantum machine, including programming the machine and computation time. It should be noted that the actual quantum computation time is at least an order of magnitude shorter than the access time (see Online document D-Wave Timing Documentation, 2024).

5.2 Discussion of the results

Quantum computation environments are already a real competition for classical supercomputer computations based on silicon processors. Especially for problems for which the natural representation of solutions are binary variables, as in the knapsack problem, translation to the QUBO model is natural and allows solving relatively large examples on the QPU quantum

Table 3 Results of computational experiments (optimal values are marked in bold)

Instance	QAdB&B			GUROBI	
	LB	UB	QPU [ms]	OPT	CPU [ms]
OR5x100−0.25_1	**39**, **109**	42, 236	16	39, 109	37
OR5x100−0.25_2	**36**, **836**	40, 401	31	36, 836	40
OR5x100−0.25_3	**34**, **279**	37, 812	31	34, 279	39
OR5x100−0.25_4	**35**, **593**	40, 050	32	35, 593	36
OR5x100−0.25_5	33, 247	36, 944	31	33, 249	45
OR5x100−0.25_6	36, 406	39, 208	16	36, 425	39
OR5x100−0.25_7	35, 954	40, 419	31	35, 969	44
OR5x100−0.25_8	**36**, **969**	40, 810	31	36, 969	41
OR5x100−0.25_9	35, 261	39, 194	15	35, 276	34
OR5x100−0.25_10	**34**, **855**	38, 935	31	34, 855	43
Average			26		39
OR30x250−0.25_1	93, 553	96, 667	16	93, 898	65
OR30x250−0.25_2	94, 620	99, 902	16	96, 320	71
OR30x250−0.25_3	90, 597	94, 109	16	91, 169	66
OR30x250−0.25_4	98, 159	101, 978	15	98, 747	64
OR30x250−0.25_5	97, 722	101, 985	15	98, 984	71
OR30x250−0.25_6	96, 262	99, 272	15	96, 475	69
OR30x250−0.25_7	89, 382	93, 048	16	90, 141	77
OR30x250−0.25_8	97, 812	101, 450	16	98, 215	69
OR30x250−0.25_9	94, 158	98, 211	16	95, 266	72
OR30x250−0.25_10	94, 905	98, 692	15	95, 706	73
Average			15		69
OR30x500−0.25_1	181, 024	187, 679	16	183, 946	204
OR30x500−0.25_2	182, 558	192, 438	16	184, 876	208
OR30x500−0.25_3	190, 230	196, 035	16	191, 857	199
OR30x500−0.25_4	184, 492	194, 754	15	190, 270	191
OR30x500−0.25_5	189, 511	199, 901	16	192, 276	164
OR30x500−0.25_6	176, 709	191, 185	16	179, 981	199
OR30x500−0.25_7	178, 692	188, 532	16	184, 622	212
OR30x500−0.25_8	182, 609	190, 747	16	185, 662	198
OR30x500−0.25_9	185, 354	193, 559	16	189, 127	210
OR30x500−0.25_10	188, 991	199, 669	15	193, 674	423
Average			15		220

processor. Comparing the quality of the solutions obtained, it should be noted that in the instances with 100 items, the quantum machine found the optimal solution in 5 out of 10 cases. The average error of the solutions, calculated as the relative difference between the value of the knapsack generated by this machine and the optimal value in relation to the optimal value, is only 0.014% for this group of instances. For instances with 250 and 500 items, this error is accordingly 0.819% and 1.969%. The average time of using a quantum

machine is comparable to the time of calculations on a classical computer for $n = 100$ and over 13 times shorter for $n = 500$.

It is worth noting that the current capabilities of quantum annealers related to the number of qubits and their connections significantly limit (to a few or a dozen tasks) the size of solvable examples of optimization and decision-making problems. This is a limitation that applies to all researchers conducting computational experiments on quantum computers. For example, in the work of Chen et al. (2022) the results are approximate for the 3-SAT problem of size $n \leq 15$, and in Jain's work Jain (2021) the largest example size for the traveling salesman problem was $n = 10$. In this context, our solving optimally most examples of size $n = 100$ and approximately for $n = 250$ and $n = 500$ represents a significant shift in the boundary of solvable problem instances. Moreover, for instances for which we have determined an approximate solution, we are able to precisely provide the maximum error of this approximation by determining the Lower and Upper Bounds (LB and UB) of the optimal solution.

6 Summary

The paper presents the potential of utilizing a quantum machine to hardware-implement quantum annealing procedures for the exact solving of the NP-hard Knapsack Problem based on the Branch and Bound method. Quantum annealing is executed on the QPU alternating with a control procedure overseeing the search tree implemented on a silicon-based CPU. The studied problem pertains to production systems, specifically focusing on production line balancing, transportation logistics, and inner-factorial route optimization. The study introduces an innovative approach, leveraging quantum machinery for hardware realization of the quantum annealing procedure, grounded in the Branch and Bound method. The results obtained from the D-Wave quantum machine indicate that, despite the probabilistic nature of quantum computations, it is possible to generate truly optimal solutions using this groundbreaking technology.

In the experimental outcomes derived from the quantum machine, a significant acceleration in solving the knapsack problem was observed compared to traditional computational methods. The integration of quantum annealing with the Branch and Bound method enabled precise and efficient control over the algorithm, leading to the discovery of optimal solutions in a shorter timeframe. Future research directions are planned to further investigate the algorithm's effectiveness and efficiency across various logistical scenarios, taking into account diverse operational conditions such as different types of heavy-duty trucks, varying load requirements, and distinct route constraints. Additionally, optimization of the quantum annealing procedure and integration with other advanced computational techniques are envisioned to enhance the algorithm's overall efficiency and performance.

Acknowledgements We gratefully acknowledge Polish high-performance computing infrastructure PLGrid (HPC Center: ACK Cyfronet AGH) for providing computer facilities and support within computational grant no. PLG/2024/017186.

Data availability The dataset for computational experiments is available at http://zasobynauki.pl/zasoby/83109. Data for a case study on the transport problem from Section 3 can be sent on request.

Declarations

Conflict of interest The authors do not declare any financial or non-financial Conflict of interest in connection with the work.

References

Ajagekar, A., Humble, T., & You, F. (2020). Quantum computing based hybrid solution strategies for large-scale discrete-continuous optimization problems. *Computers & Chemical Engineering, 132*, 106630.

Barakat, D.E.-S., El-Henawy, I. M., & Abdel-Basset, M. (2016). Knapsack Problems: a comprehensive review. *Asian Journal of Mathematics and Computer Research, 14*(4), 296–317.

Biglar, A. (2018). *Some applications of Knapsack problem*, (2018). https://doi.org/10.13140/RG.2.2.15115.39209.

Bożejko, W., Pempera, J., Uchroński, M., & Wodecki, M. (2022). Distributed Quantum Annealing on D-Wave for the Single Machine Total Weighted Tardiness Scheduling Problem. In D. Groen, C. de Mulatier, M. Paszynski, V.V. Krzhizhanovskaya, J.J. Dongarra, P.M.A. Sloot, (Ed.), *Computational Science*, 13353, (pp. 171–178). LNCS.

Bożejko, W., Pempera, J., Uchroński, M., & Wodecki, M. (2024). Quantum annealing-driven branch and bound for the single machine total weighted number of tardy jobs scheduling problem. *Future Generation Computer Systems, 155*, 245–255.

Bożejko, W., Klempous, R., Pempera, J., Rozenblit, J., Uchroński, M., & Wodecki, M. (2023). Optimal solving of a scheduling problem using quantum annealing metaheuristics on the D-Wave quantum solver, TechRxiv, Preprint, https://doi.org/10.36227/techrxiv.22677721.v1.

Cacchiani, V., Iori, M., Locatelli, A., & Martello, S. (2022). Knapsack problems - An overview of recent advances. Part I: Single knapsack problems. *Computers & Operations Research, 143*, 105692.

Camargo, V., Mattiolli, L., & Toledo, F. (2012). A knapsack problem as a tool to solve the production planning problem in small foundries. *Computers & Operations Research, 39*, 86–92.

Chen, Y. Q., Chen, Y., Lee, C. K., Zhang, S., & Hs, C. Y. (2022). Optimizing quantum annealing schedules with Monte Carlo tree search enhanced with neural networks. *Nature Machine Intelligence, 4*, 269–278.

Cho, M. (2019). The Knapsack Problem and Its Applications to the Cargo Loading Problem, *Analysis of Applied Mathematics*, *48*.

Coniglio, S., Furini, F., & San Segundo, P. (2021). A new combinatorial branch-and-bound algorithm for the Knapsack Problem with Conflicts. *European Journal of Operational Research, 289*(2), 435–455.

Dam, W., Eldefrawy, K., Genise, N., & Parham, N. (2021). Quantum Optimization Heuristics with an Application to Knapsack Problems. *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, (pp. 160–170) Broomfield, CO, USA, (2021).

Denkena, B., Schinkel, F., Pirnay, J., & Wilmsmeier, S. (2021). Quantum algorithms for process parallel flexible job shop scheduling. *CIRP Journal of Manufacturing Science and Technology, 33*, 100–114.

He, Y., Wang, J., Liu, X., Wang, X., & Ouyang, H. (2024). Modeling and solving of knapsack problem with setup based on evolutionary algorithm. *Mathematics and Computers in Simulation, 219*, 378–403.

Jain, S. (2021). Solving the Traveling Salesman Problem on the D-Wave Quantum Computer. *Frontiers in Physics, 9*, 760783. https://doi.org/10.3389/fphy.2021.760783

Ji, Z., Wang, Z., Wu, T., & Huang, W. (2017). Solving the 0–1 knapsack problem based on a parallel intelligent molecular computing model system. *Journal of Intelligent & Fuzzy Systems, 33*(5), 2719–2726.

Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*. Berlin: Springer.

Laabadi, S., Naimi, M., El Amri, H. & Achchab, B. (2018). The 0/1 Multidimensional Knapsack Problem and Its Variants: A Survey of Practical Models and Heuristic Approaches. *American Journal of Operations Research*, *8*(05), https://doi.org/10.4236/ajor.2018.85023.

Li, X., Fang, W., & Zhu, S. (2023). An improved binary quantum-behaved particle swarm optimization algorithm for knapsack problems. *Information Sciences, 648*, 119529.
Online document D-Wave Timing Documentation. Retrieved April 7, 2024. https://docs.dwavesys.com/docs/latest/c_qpu_timing.html
Pempera, J., & Uchroński, M. (2023). *Benchmark test files for the binary knapsack problem*, http://zasobynauki.pl/zasoby/83109.
Pusey-Nazzaro, L., & Date, P. (2020). Adiabatic Quantum Optimization Fails to Solve the Knapsack Problem, arxiv:2008.07456.
Refaei, R., Zhang, A. Y., Firat, M., & Kaymak, U. (2020). A State Aggregation Approach for Solving Knapsack Problem with Deep Reinforcement Learning. *Proceedings of Machine Learning Research, 129*, 81–96.
Rizk-Allah, R. M., & Hassanien, A. E. (2018). New binary bat algorithm for solving 0–1 knapsack problem. *Complex & Intelligent Systems, 4*, 31–53.
Shen, J., Shigeoka, K., Ino, F., & Hagihara, K. (2019). GPU-based branch-and-bound method to solve large 0–1 knapsack problems with data-centric strategies. *Concurrency and Computation: Practice and Experience, 31*, e4954.
Vasilchikov, V. V. (2018). On a Recursive-Parallel Algorithm for Solving the Knapsack Problem. *Automatic Control and Computer Sciences, 52*, 810–816.
Vu, T., & Derbel, B. (2016). Parallel Branch-and-Bound in multi-core multi-CPU multi-GPU heterogeneous environments. *Future Generation Computer Systems, 56*, 95–109.
Wang, Z., Zhang, H., & Li, Y. (2022). Fast Polynomial Time Approximate Solution for 0-1 Knapsack Problem. *Hindawi Computational Intelligence and Neuroscience*, Article ID 1266529, https://doi.org/10.1155/2022/1266529.
Wilbaut, Ch., Hanafi, S., & Salhi, S. (2008). A survey of effective heuristics and their application to a variety of knapsack problems. *IMA Journal of Management Mathematics, 19*(3), 227–244.
Yarkoni, S., Raponi, E., Bäck, T., & Schmitt, S. (2022). Quantum annealing for industry applications: introduction and review. *Reports on Progress in Physics, 85*(10), 104001. https://doi.org/10.1088/1361-6633/ac8c54
Zavala-Diaz, J. C., Cruz-Chávez, M. A., López-Calderón, J., Hernández-Aguilar, J. A., & Luna-Ortíz, M. E. (2019). A Multi-Branch-and-Bound Binary Parallel Algorithm to Solve the Knapsack Problem 0–1in a Multicore Cluster. *Applied Sciences, 9*, 53–68.
Zhang, J. (2011). Comparative Study of Several Intelligent Algorithms for Knapsack Problem. *Procedia Environmental Sciences, 11*, 163–168.