Hochschule
Bonn-Rhein-Sieg
University
of Applied Sciences

Fachbereich Informatik
Department of Computer Sciences

Forschungszentrum
Jülich GmbH
Jülich Supercomputing
Centre

Computer Science, Bachelor of Science
spez. in complex software systems

Comparing classical and quantum-based approaches for solving the 0/1 knapsack problem

by

Marvin Schönwälder

Examined by

Prof. Dr. Ralf Thiele, Hochschule Bonn-Rhein-Sieg
Dr. Dennis Willsch, Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH

Name:
Adresse:

Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Unterschrift Sankt Augustin, den

Contents

Acronyms

BQM	Binary Quadratic Model
CFA	Controlled Full Adder
DP	Dynamic Programming
FA	Full Adder
HA	Half Adder
KP	0/1 Knapsack Problem
LSB	Least Significant Bit
MSB	Most Significant Bit
QA	Quantum Annealing
QPU	Quantum Processing Unit
QUBO	Quadratic Unconstrained Binary Optimization
$\mathbf{TTS}_{99}$	Time to Solution 99%

1 Introduction

"I can't find an efficient algorithm, I guess I'm just too dumb." (Garey and Johnson, 1979). This iconic quote captures the frustration of encountering problems that seem to defy efficient computation. However, this frustration is not solely a theoretical concern. Optimization problems are fundamental to nearly every aspect of practical application. From logistics and resource allocation to finance and manufacturing, the ability to find optimal solutions to complex problems directly impacts efficiency, profitability, and performance. Many of these problems are categorized as $\mathcal{NP}$-hard, a class of problems for which no efficient polynomial-time solution is believed to exist. The opening quote hints at exactly this. The obstacle is not the person, but the complexity of the task itself. The Knapsack Problem is a prime example of such a notoriously hard optimization problem.

The central research question of this thesis is how classical and quantum-based approaches differ when solving the 0/1 Knapsack Problem (KP) with respect to runtime, resource consumption, and solution quality. A key focus of this thesis lies on a newly proposed Controlled Full Adder (CFA) approach for encoding the KP on quantum annealers. Such an encoding for the KP has not appeared in literature before.

To answer the research question, three hypotheses are formulated in Section 1.1. H1 and H2 act as baselines for the well-known classical and slack-based encodings, while H3 is the central hypothesis that this thesis examines in greater depth.

1.1 Hypotheses

The following hypotheses address the three comparison dimensions of runtime, resource consumption, and solution quality, covering both the classical and the quantum-based approaches.

H1 (classical baseline). Dynamic programming with tabulation (DP) is expected to outperform a brute-force search in runtime for instances with moderate capacities. However, because its runtime is pseudo-polynomial, its performance may degrade for very large capacities.

H2 (quantum baseline). The slack-based Quadratic Unconstrained Binary Optimization (QUBO) encoding is expected to provide a valid, practically usable formulation for quantum annealing. For small to medium-sized instances it should often produce optimal solutions. Its success frequency is expected to decline for larger instances.

H3 (central hypothesis). The CFA QUBO is hypothesized to be a technically feasible encoding that improves solution quality, measured as the frequency of finding optimal solutions, compared with the slack-based encoding. This potential improvement is expected to require substantially more physical qubits, which consequently increases the embedding complexity.

1.2 Implementation Strategies

The implementation strategies in this thesis are chosen to support the CFA approach as the main subject while still enabling a structured comparison with established reference methods. On the classical side, a brute-force solver and a DP solver are implemented as exact reference solvers. On the quantum side, two QUBO encodings are implemented. The first is the established slack-based formulation, which serves as the baseline for quantum annealing. The second is the CFA formulation, the main alternative explored in this thesis. All implementations use the same benchmark instances and a uniform output format, so that runtime, resource consumption, and solution quality can be compared consistently across approaches.

2 Background

This chapter provides the background for the rest of the thesis in two parts. The technical background covers the quantum-computing concepts behind quantum annealing and the D-Wave hardware used in the experiments. The theoretical background then introduces the knapsack problem, the classical algorithms used as references, and the slack-based QUBO formulation.

2.1 Technical Background

The technical background of this thesis starts by introducing the quantum-computing concepts required to understand how quantum-based computation works. This section covers the basic principles of quantum annealing, the Ising and QUBO formulations, and the D-Wave hardware platform on which the quantum experiments are carried out.

2.1.1 Quantum Computing

It remains a challenging task to find an optimal solution to $\mathcal{NP}$-hard problems using classical computers. Quantum computers are not expected to solve this problem; however, they rely on a computational model that fundamentally differs from classical computation (Preskill, 2018). This makes them an interesting alternative for certain optimization problems, including the KP.

The two main quantum-computing paradigms are gate-based quantum computers and quantum annealers. The latter is specifically designed for optimization problems. Since the KP is an optimization problem, quantum annealers are particularly well suited for solving it. Therefore, quantum annealing (QA) will be the main focus of this thesis. All quantum-based approaches will be implemented and evaluated on a D-Wave quantum annealer, which is one of the leading platforms for QA.

Every quantum computer uses qubits for carrying information. Qubits can be realized physically in various ways, such as a single electron, a single photon or a very cold superconducting electrical circuit (Preskill, 2018). The state of a qubit is modeled within a two-dimensional complex vector space. A qubit can be mathematically described as a superposition of the two basis states $|0\rangle$ and $|1\rangle$, which can be represented as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} \quad \text{with} \quad \alpha, \beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1. \tag{1}$$

When a qubit is in a superposition, it means that it is not restricted to be in one of the two basis states, but can be in a combination of both states simultaneously. In the broader quantum-computing context, this property is one of the reasons quantum systems can explore multiple computational

states at once. This does not directly imply classical-style parallel processing in quantum annealing.

To solve a problem on a quantum annealer, the problem has to be reformulated in a specific way. It either has to be encoded into an Ising model or a QUBO model. Both formulations are mathematically equivalent, although they differ in the way they represent the problem variables. The Ising model is defined over spin variables $s_i \in \{-1, +1\}$ (Lucas, 2014), whereas the QUBO model represents the qubit states through binary variables $x_i \in \{0, 1\}$ (Prunnen, 2022).

The Ising model is typically expressed as an energy function, which is given by

$$E(s) = \sum_{i=1}^{n} h_i s_i + \sum_{i=1}^{n} \sum_{j=1}^{i-1} J_{ij} s_i s_j \tag{2}$$

with biases h_i and couplers J_{ij}, while the QUBO model is expressed as

$$E(x) = \sum_{i=1}^{n} a_i x_i + \sum_{i<j}^{n} b_{ij} x_i x_j = \sum_{i \leq j}^{n} x_i Q_{ij} x_j. \tag{3}$$

In this formulation, the linear coefficients of the QUBO are represented by the diagonal entries Q_{ii} of the coefficient matrix Q while the quadratic coefficients are represented by the off-diagonal entries Q_{ij} (Prunnen, 2022). When converting from Ising to QUBO or vice versa, the relationship between the coefficients is given by $h_i = \frac{a_i}{2} + \sum_{j \neq i} \frac{b_{ij}+b_{ji}}{4}$ and $J_{ij} = \frac{b_{ij}}{4}$. Since minimizing the Ising energy function is $\mathcal{NP}$-complete in its decision version, every $\mathcal{NP}$ problem can be reduced to an Ising formulation (Lucas, 2014).

In physics, a quantum system is described in terms of a so-called Hamiltonian, which is a matrix that produces the time evolution of a system. QA aims to reach the lowest-energy configuration, the ground state, of a given Ising Hamiltonian. This Ising Hamiltonian is defined as

$$H_{\mathrm{P}} = \sum_{i=1}^{N} h_i \sigma_z^{(i)} + \sum_{i=1}^{N} \sum_{j=1}^{i-1} J_{ij} \sigma_z^{(i)} \otimes \sigma_z^{(j)} \tag{4}$$

where σ_z is the Pauli-Z matrix acting on the i-th qubit. When an optimization problem is converted into an Ising model, its optimal solution is encoded into this ground state of H_{P}. H_{P} is derived from Eq. 2, where s_i and s_j are replaced by $\sigma_z^{(i)}$ and $\sigma_z^{(j)}$ respectively to represent the quantum state of the i-th and j-th qubits.

The process of QA relies on the adiabatic theorem. It states that a system that starts in the ground state of a Hamiltonian and progresses slowly enough, will remain in the instantaneous ground state throughout the evolution. This starting Hamiltonian is the driver Hamiltonian H_{D}, for

which the system is guided toward the problem Hamiltonian H_{P}. The total evolution is described by the Hamiltonian $H(s)$:

$$H(s) = A(s)H_{\mathrm{D}} + B(s)H_{\mathrm{P}} \tag{5}$$

where $s \in [0, 1]$ represents the normalized annealing time where $s = 0$ is the start and $s = 1$ is the end of the annealing process. The functions $A(s)$ and $B(s)$ define the annealing schedule, where $A(s)$ commonly decreases to zero and $B(s)$ increases from zero over the interval.

The standard driver Hamiltonian the system is initialized with is defined as

$$H_{\mathrm{D}} = -\sum_{i=1}^{N} \sigma_x^{(i)}, \tag{6}$$

where σ_x is the Pauli-X matrix acting on the i-th qubit. The ground state of the driver Hamiltonian is an equal superposition of all computational basis states, enabling the system to initially explore the entire solution space. If the evolution is sufficiently slow and the minimum energy gap between the ground state and the first excited state does not become too small, the system reaches the optimal solution with high probability (Adame and McMahon, 2018).

In a physical system, the control functions $A(s)$ and $B(s)$ are implemented as time-dependent bias and coupling strengths unique to the hardware's used quantum processing unit (QPU). As shown in Figure 1, the D-Wave Advantage2_system2 has a non-linear annealing schedule.

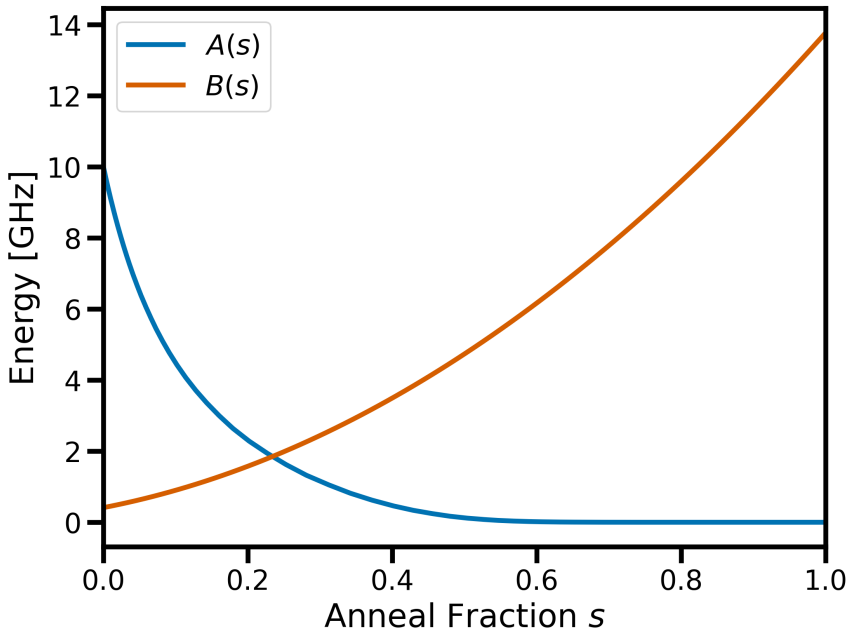

Figure 1: Annealing schedule of the D-Wave Advantage2_system2, showing how the control functions $A(s)$ and $B(s)$ vary over the normalized annealing time, gradually shifting from the driver to the problem Hamiltonian.

However, there are some practical limitations. Although the adiabatic theorem provides a theoretical guarantee for finding the optimal solution, it is not always achievable in practice (Aharonov et al., 2005). This is due to various factors such as noise, a limited annealing time, and hardware limitations. Another obstacle is the need for ancillary qubits to encode constraints and higher-order terms in the Ising formulation. A further constraint is the hardware's limited connectivity. When mapping a problem to the hardware graph, a single logical variable may need to be represented by multiple physical qubits to create the necessary couplings. As a result, the number of required qubits grows, creating an ancilla overhead that further limits the problem size that can be embedded on the annealing hardware (Imoto et al., 2025).

2.1.2 D-Wave Quantum Annealer

D-Wave is a company that has specialized in QA and has developed several generations of quantum annealers. The newest generation is the Advantage2. In this thesis, all problems solved on a quantum annealer are solved on the D-Wave Advantage2_system2 located in Jülich, Germany. The QPU provides 4516 qubits and 40448 couplers, which are arranged in a Zephyr topology (D-Wave Quantum Inc., 2026b). This graph has 20 couplers connecting each

qubit to 20 other qubits on average, as shown in Figure 2 (D-Wave Quantum Inc., 2026c).

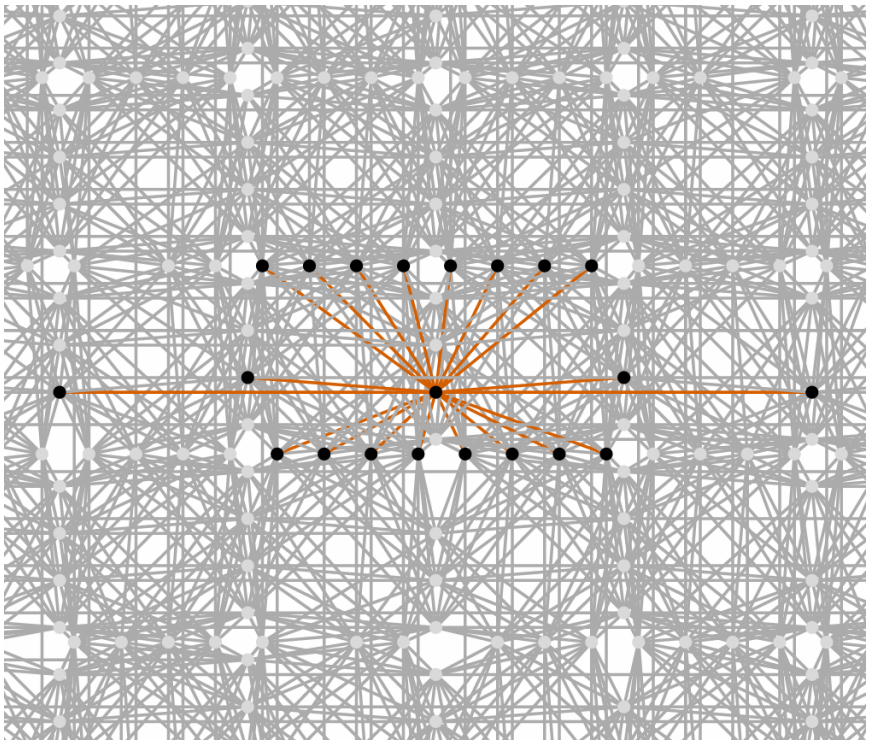

Figure 2: Zephyr connectivity graph of the D-Wave Advantage2_system2, showing one qubit and its couplings to neighboring qubits in the physical hardware graph.

To solve a problem on the D-Wave quantum annealer, a formulation as QUBO or Ising must be embedded into the hardware graph of the QPU. The annealer internally uses only Ising formulations. However, the D-Wave software stack also supports QUBO formulations, which are automatically converted into an Ising formulation before being processed. The converted problem is then embedded into the physical Zephyr graph of the QPU. Since finding an optimal embedding is itself an $\mathcal{NP}$-hard problem, this step is performed with a heuristic method. Embedding is a crucial step, as its quality directly influences the annealer's performance. In particular, limited hardware connectivity can force a single logical variable to be represented by a chain of multiple physical qubits, which makes the process computationally intensive and increasingly difficult for highly connected problems. Suboptimal embeddings can introduce additional noise and thermal fluctuations, thereby degrading solution quality. Furthermore, since physical qubits sometimes develop faults, the annealer has to be calibrated regularly to identify and disable faulty qubits. This leads to a non-uniform distribution of active

qubits across the QPU over time, which complicates the embedding process further.

2.2 Theoretical Background

This section defines the knapsack problem formally, summarizes the classical solution methods, and derives the slack-based QUBO encoding.

2.2.1 0/1 Knapsack Problem

A variety of real-world decision problems are more similar than they first appear. For instance, a mountaineer deciding which items to pack into their knapsack, an investor choosing which investments to make within a finite budget, and a student selecting which questions to answer within a limited timeframe. At first glance, these situations appear entirely distinct. Yet they share the same fundamental structure. Each problem involves making binary decisions. Such a choice can be modeled by a binary variable $x \in \{0, 1\}$, where $x = 1$ indicates that an option is selected and $x = 0$ that it is not selected (Kellerer et al., 2004).

Consider a problem with n such variables, where each item i is associated with a profit value $v_i \geq 0$ and a weight $w_i \geq 0$. The total profit of a selection is given by

$$\sum_{i=1}^{n} v_i x_i,$$

while the total weight is given by

$$\sum_{i=1}^{n} w_i x_i.$$

A feasible solution must satisfy a capacity constraint

$$\sum_{i=1}^{n} w_i x_i \leq W,$$

where W denotes the maximum capacity of the knapsack. The resulting optimization problem is the KP, which can be formally defined as

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} v_i x_i \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0, 1\}, \quad i = 1, \ldots, n \end{aligned} \tag{7}$$

(Kellerer et al., 2004). Note that the notation differs from Kellerer et al. (2004), where items are indexed by j instead of i, the profits are denoted by

p_j instead of v_i, and the capacity is denoted by c instead of W. However, the mathematical formulation is equivalent.

One of the main goals of studying the KP is the development of algorithms that compute optimal or approximate solutions. The performance of such algorithms can be evaluated in several ways. The first approach is to implement the algorithm and measure its performance on several different test instances. However, results strongly depend on hardware and implementation quality. The second approach is an average-case analysis. Yet it is often unclear which notion of "average" is appropriate for real-world instances. Worst-case analysis is the third common approach to evaluate an algorithm. It provides an asymptotic upper bound on the running time as a function of the input size. This bound is typically expressed using Big-$\mathcal{O}$ notation and characterizes the maximum number of arithmetic operations required to solve any instance of a given size.

For example, a running time of $\mathcal{O}(n^2)$ means that the maximum number of required computational steps grows proportionally to the square of the number of items n. This runtime depends only on the input size and therefore provides a clear asymptotic bound. However, some algorithms have a pseudo-polynomial runtime. This means that the runtime depends not only on n, but also on the numerical values of the input parameters. For the KP, DP is a typical example, as its runtime of $\mathcal{O}(nW)$ depends not only on n but also on the capacity W (Cormen et al., 2022).

The KP is one of the best-known and most widely studied $\mathcal{NP}$-hard optimization problems. Despite its simple formulation, no polynomial-time algorithm is known that solves all instances. If such an algorithm existed, a polynomial-time solution for the $\mathcal{NP}$-hard optimization version of the KP would also provide a polynomial-time solution for its corresponding decision problem. As the KP's decision version is $\mathcal{NP}$-complete, $\mathcal{NP}$-completeness theory implies that a polynomial-time algorithm for it would make every problem in $\mathcal{NP}$ solvable in polynomial time. This outcome is widely believed to be unlikely (Karp, 1972).

2.2.2 Solving the 0/1 Knapsack Problem Classically

To solve the KP classically, various algorithms have been proposed to be effective. Some of these are exact algorithms that guarantee finding the optimal solution. Other, so-called heuristic algorithms, intend to find near-optimal and sometimes even optimal solutions in polynomial time. While sacrificing optimality, heuristic algorithms are able to provide a more time-efficient solution.

In this thesis, the focus for classical algorithms is on exact algorithms, as they are more suitable for benchmarking the performance of quantum annealers due to their ability to guarantee optimal solutions. However, heuristic algorithms remain important in practice, since they can quickly

produce near-optimal solutions for large instances where exact methods become infeasible.

Brute Force The most straightforward way to solve the KP is to use a brute-force approach. It is based on enumerating all 2^n possible combinations of items and checking which one yields the maximum profit while satisfying the capacity constraint. This approach is simple to implement and would in principle always find the optimal solution (Cormen et al., 2022). However, it is computationally infeasible when the instances grow large, as checking 2^n combinations leads to an exponential time complexity of $\mathcal{O}(2^n)$.

Dynamic Programming with Tabulation Another way to solve the KP classically is to use DP. The idea is to break the problem into smaller subproblems and store their solutions in a table, which avoids redundant calculations and reduces the time complexity from exponential to pseudo-polynomial. The construction relies on the Bellman recursion, which defines the best achievable value of a problem as a composition of the solutions to its subproblems (Bellman, 1957). This follows the principle that an optimal global solution can be assembled from the optimal solutions of its subproblems (Cormen et al., 2022). A single pass through the table is sufficient, giving a pseudo-polynomial time complexity of $\mathcal{O}(nW)$ (Kellerer et al., 2004).

2.2.3 Slack-based QUBO

Deriving the QUBO coefficient matrix directly from the mathematical formulation of the optimization problem is not always straightforward or unique. In practice, a QUBO formulation is constructed as an energy function that combines an objective term with one or more penalty terms that represent the problem's constraints. Each penalty term is weighted by a parameter λ, which sets the strength of the penalty. The resulting energy function only contains linear and quadratic terms.

Optimization problems such as the KP typically have an inequality constraint. These cannot be encoded directly into a QUBO, as the QUBO formulation only allows for unconstrained optimization problems. The first step in deriving a QUBO for the KP is to transform the inequality constraint into an equality constraint. An auxiliary slack variable S is introduced to make the penalization term disappear when the constraint is satisfied. To represent S in binary, $N = \lfloor \log_2(W) \rfloor + 1$ binary variables are required. The slack variable can be expressed as

$$S = \sum_{k=0}^{N-1} 2^k s_k. \tag{8}$$

When introducing the slack variable, the resulting QUBO will contain more qubits than the original problem. This ultimately reduces the size of the problem that can be solved on a quantum computer (Montañez-Barrera et al., 2023). It should be noted that slack-based approaches are commonly used, but are not the only way of transforming optimization problems into a QUBO. Some alternative formulations have been proposed by Montañez-Barrera et al. (2023).

Converting 0/1 Knapsack to QUBO with Slack Variables Following the steps from Section 2.2.3, the KP can be transformed into a QUBO by first introducing the slack variable S:

$$\begin{aligned} &\text{maximize} \quad \sum_{i=1}^{n} v_i x_i \\ &\text{subject to} \quad \sum_{i=1}^{n} w_i x_i + \sum_{k=0}^{N-1} 2^k s_k = W, \quad x_i \in \{0,1\}, \quad i = 1, \ldots, n. \end{aligned} \tag{9}$$

After this, the profit function needs to be adjusted to represent the highest value as a state of the lowest energy. Finally, the new equality constraint is turned into a quadratic penalty term that enforces the original capacity constraint. Adding this penalty to the energy function results in

$$E(x,s) = -\sum_{i=1}^{n} v_i x_i + \lambda(\sum_{i=1}^{n} w_i x_i + \sum_{k=0}^{N-1} 2^k s_k - W)^2$$

(Quintero and Zuluaga, 2021). Therefore, $E(x,s)$ is the energy function that is minimized by the quantum annealer.

Finding the right value for λ is a crucial step for finding good solutions. It must be large enough so that any violation of the capacity constraint is penalized sufficiently. In other words, every infeasible solution must have a higher energy than any feasible optimal solution. At the same time, it should be as small as possible to avoid unnecessarily large coefficients in the QUBO, which would lead to a worse performance (see Section 3.1). A good baseline for this approach is $\lambda = \max\{v_1, \ldots, v_n\}$, as finding an optimal λ is a research question in its own right (Quintero and Zuluaga, 2021).

3 Methodology

All approaches are implemented in Python within Jupyter notebooks.[1] While the classical algorithms are run locally, the quantum-based approaches are carried out using the D-Wave Ocean SDK, which contains routines to embed and sample QUBO and Ising problems on quantum annealers.

Apart from the algorithms, the rest of the notebooks contain helper routines to import the test dataset, benchmark the approaches, visualize the resulting data and save the results and the plots. Additionally, another notebook is implemented to generate the test dataset. The dataset is generated with systematically varying properties as further described later in the evaluation strategy. A python script is implemented to generate a comparison dataset with all optimal solutions for each test instance.

3.1 Controlled Full Adder

The slack-based formulation in Section 2.2.3 provides a straightforward way to encode the KP into a QUBO, but it has one important drawback. The binary encoding of the slack variable creates large coefficients that can reach values up to $2^N - 1$, which can lead to performance issues. The coefficients of the Ising formulation have to lie inside a certain range to be processed by the D-Wave quantum annealer. In particular, the linear coefficients are restricted to $h_i \in [-4.0, 4.0]$ and the quadratic coefficients to $J_{i,j} \in [-1.0, 1.0]$ (D-Wave Quantum Inc., 2026a). If the coefficients grow larger, each of them has to be scaled down to fit within this range. This can lead to accuracy problems, as the relative differences between the coefficients may be lost during scaling, specifically when the scaled difference falls below the machine's precision.

To avoid these large coefficients, an alternative approach maps classical logic and arithmetic circuits directly onto the hardware instead of using a single penalty term. Expressing the capacity constraint through many small local circuits keeps the individual coefficients small, rather than letting them grow with the problem size. This idea is not new and has been used in other settings, though not for the KP. For SAT, cardinality constraints have been encoded with counter circuits, including a parallel counter built from half adders and full adders (HA and FA). These encodings are expressed as CNF clauses for classical SAT solvers (Sinz, 2005). For integer factoring, adder circuits have been mapped onto a quantum annealer to implement the binary multiplication of the two unknown factors (Willsch et al., 2025). The KP differs from both. It neither counts bits, as in the SAT cardinality encoding, nor multiplies two unknowns, as in factoring. Instead, it sums the weighted contributions $w_i x_i$ of the selected items, where each known weight w_i is added through a single selection bit x_i. Applying an adder-based encoding to the KP in this way is therefore a new idea.

[1] Source code: `https://github.com/MvnSwr/Bachelor-Thesis/releases/tag/v1.0`

For the KP, the idea is to construct a circuit from HAs and FAs whose inputs correspond to the weights of the individual items and which computes the total weight of the packed knapsack in binary representation. To enforce the capacity constraint in this adder-based representation, the central idea is to detect infeasible total weights directly in the calculated binary output. If the capacity has the form $2^k - 1$ with $k \in \mathbb{N}$, every output variable above this range can be interpreted as an overflow bit. Hence, a violation of the capacity constraint is represented when an overflow variable in $Z_k, Z_{k+1}, \ldots$ is set to 1, which can then be penalized. Since instances do not always satisfy this structural form of the capacity, an equivalent transformed instance with an adapted capacity is used. Conceptually, this keeps the original optimization target while making the overflow interpretation of the output bits directly usable for the penalization. When adjusting the capacity to the necessary form of $2^k - 1$, mandatory zero-value dummy items are added to regain the original optimization target. The concrete realization of this transformation is described in Section 4.3.

Based on this constraint interpretation, the adder circuit is built from the binary representations of the item weights and the knapsack capacity. As shown in Figure A.1, each item weight is represented bitwise by variables $V_{i,j}$, where i denotes the item index and j the bit position from the least significant bit (LSB) to the most significant bit (MSB). To keep the adder structure as small as possible, only the $V_{i,j}$ whose weight bits could actually be set to 1 when the item is chosen are added to the circuit. All unnecessary $V_{i,j}$, which correspond to bits that are always zero, are left out, as they do not contribute to the final sum. The item-selection variable x_i (one qubit per item) controls the corresponding weight bits $V_{i,j}$ and determines whether the $V_{i,j}$ are set.

Although the CFA approach is technically more complex than the slack-based formulation, it offers two advantages. It avoids the large coefficients introduced by the slack variables, and because the adders only connect neighboring bit positions, it relies mostly on local connectivity instead of the dense, all-to-all coupling required by the slack-based QUBO. Both properties may improve solution quality and embeddability on the D-Wave quantum annealer.

3.2 Evaluation Strategy

The empirical evaluation of each approach is based on a generated benchmark dataset of test instances. The results are analyzed both graphically and comparatively. This captures the absolute performance of each method but also its relative behavior across problem sizes. The comparison is divided into an intra-paradigmatic and an inter-paradigmatic part. The intra-paradigmatic comparison examines approaches within the same computational paradigm, i.e. classical versus classical and quantum versus quantum. The inter-paradigmatic comparison analyzes the differences between classical and quantum approaches while accounting for the fundamental differences between the paradigms. This is carried out in three steps: a statistical comparison of the metrics across all approaches, a validation of the hypotheses H1–H3 against the empirical results, and a discussion of the practical implications and methodological limitations.

3.2.1 Dataset and Benchmarking Setup

A structured dataset of test instances is generated to cover a broad range of problem sizes and instance characteristics. It has the following properties:

- Number of items n: 2–25 items.
- Item values v_i: randomly generated in the interval $[1, 2^b - 1]$, where $b \in \{2, 4, 6, 8\}$ denotes the bit width.
- Item weights w_i: randomly generated in the interval $[1, 2^b - 1]$, where $b \in \{2, 4, 6, 8\}$ denotes the bit width.
- Knapsack capacity W: approximately 30% of the total item weight, with a deviation of $\pm 10\%$. Therefore, W is also dependent on the random generation of the item weights w_i and its upper bound denoted by b.

Random generation uses a fixed, incrementing seed to ensure reproducibility. For each problem size $n \in [2, 25]$, exactly four distinct instances are generated, resulting in a total of 96 instances. Based on the bit width b, the dataset splits into four sub-datasets with different value and weight ranges, which allows the impact of the numerical range on performance to be analyzed.

All classical benchmarks are executed locally on a dedicated machine with an Intel Core i7-8700K @ 5.0 GHz and 16 GB RAM running Windows 11. Each instance is run ten times per solver to reduce the impact of background processes and runtime variability, preceded by an unmeasured warm-up run per solver to limit setup overhead. The reported runtime and memory usage are the arithmetic mean across these repetitions.

For the quantum approaches, each instance is embedded once and then sampled 1000 times. This provides a robust statistical basis for evaluating

the success frequency of the annealer. A single embedding is used to avoid the chain-strength bias that arises when several embeddings share the QPU simultaneously. Mapping the same instance multiple times onto different regions results in varying chain lengths. A global `chain_strength` is applied to the entire QPU during sampling and the embedding with the longest chains dictates the `chain_strength` for all of them. Applying this larger value to the more compact embeddings changes their energy landscape, which disproportionately degrades their solution quality and biases the results.

3.2.2 Metrics and Evaluation Criteria

Three metrics are used across all approaches:

- **Runtime:** For the classical algorithms, runtime is the measured execution time. For the quantum approaches, a single annealing run is probabilistic and does not guarantee an optimal solution, so runtime is expressed through the Time-to-Solution at 99% confidence (TTS_{99}). It is the expected total execution time required to observe an optimal solution at least once with a certainty of 99%, defined as

$$TTS_{99} = \frac{\ln(1 - 0.99)}{\ln(1 - p_{\text{sol}})} T_A, \tag{10}$$

  where p_{sol} is the empirical success frequency, i.e. the fraction of samples that yield an optimal solution. T_A is the total hardware time that covers the complete operational cycle, combining the annealing time, the state-readout time and the electronic delay between successive runs (Mehta et al., 2022).

- **Resource consumption.** For the classical algorithms, the peak memory consumption is measured. For the quantum approaches, the count of physical qubits occupied by the embedding is recorded instead.

- **Solution quality.** The classical solvers are exact and therefore always return an optimal solution. Their solution quality is guaranteed. For the quantum approaches, solution quality is captured by the success frequency p_{sol} introduced above. It represents the fraction of the 1000 samples whose item-selection variables x_i match an optimal solution. In this case, an optimal solution denotes a combination of items that fills the knapsack optimally. It does not necessarily correspond to the ground state of the full QUBO as the auxiliary variables (slack or adder bits) are not considered in this check.

4 Implementation

This chapter describes how the four solvers are implemented in Python: the two classical algorithms and the two quantum-based QUBO encodings. The quantum approaches are built on the D-Wave Ocean stack, where the notebooks do not operate on a QUBO directly. Instead, the underlying QUBO formulation is represented as a `BinaryQuadraticModel` (BQM), the standard data structure of the Ocean stack. In this thesis, therefore, QUBO and Ising refer to the mathematical problem formulations, while BQM denotes their technical representation in code. For readability, the pseudocode describes this step as assigning the formulation to a BQM before it is passed to the sampler.

4.1 Implementation of the Classical Approaches

To provide a robust baseline for evaluating the quantum solvers, two classical approaches for solving the KP are implemented. A naive brute-force search and a DP algorithm. Both are implemented as separate functions with a uniform signature. They take the lists of item values and weights along with the knapsack capacity as input, and return a standardized output tuple containing the maximum achievable value, the total weight of the selected items, and a list of their indices. This uniform interface allows the solvers to be passed as callable arguments to the same benchmarking routine (Section 3.2.1). This structured and modular design makes it easy to extend with further solvers. During benchmarking, the runtime is measured with `time.perf_counter` in microseconds (μs), and the peak memory consumption is recorded with the `tracemalloc` library.

4.1.1 Brute Force

The brute-force solver aims to find the optimal solution by exploring the entire solution space. It recursively builds a binary decision tree representing all possible combinations of items. At each level, the algorithm branches into two cases for the current item. Either including it in the knapsack or excluding it. The algorithm then invokes itself for the remaining items. The base case of the recursion is reached when all items have been evaluated or the remaining capacity drops to zero.

To improve efficiency and avoid exploring infeasible solutions, the algorithm implements feasibility pruning. Before a branch includes an item, it checks whether the item's weight exceeds the remaining capacity. If including the item would violate the capacity constraint, that branch is pruned and the algorithm only proceeds with the exclusion branch. This significantly reduces the decision tree and the number of combinations that need to be evaluated. Although the worst-case time complexity remains exponential at $\mathcal{O}(2^n)$, this

optimization provides practical speedups. Especially for the test instances used in this thesis which have a relatively small capacity-to-total-weight ratio. By tracking the most valuable feasible combination as the recursion unwinds, the solver is guaranteed to return an optimal subset.

4.1.2 Dynamic Programming with Tabulation

for $w \leftarrow 0$ **to** W **do**
  $table[0, w] \leftarrow 0$;
end
for $i \leftarrow 1$ **to** n **do**
  for $w \leftarrow 0$ **to** W **do**
    if $w_{i-1} \leq w$ **then**
      $table[i, w] \leftarrow \max(table[i-1, w], v_{i-1} + table[i-1, w-w_{i-1}])$;
    end
    else
      $table[i, w] \leftarrow table[i-1, w]$;
    end
  end
end
$items \leftarrow [\,]$;
$w \leftarrow W$;
$i \leftarrow n$;
while $i \geq 1$ **do**
  if $table[i, w] \neq table[i-1, w]$ **then**
    $items$.append$(i-1)$;
    $w \leftarrow w - w_{i-1}$;
  end
  $i \leftarrow i - 1$;
end
$bestValue \leftarrow table[n, W]$;
$usedWeight \leftarrow \sum_{i \in items} w_i$;
return $bestValue$, $usedWeight$, $items$;

Algorithm 1: Tabulation scheme for the dynamic-programming KP solver, showing the table initialization, the row-wise Bellman recursion, the backtracking that recovers the selected items, and the returned solution.

The DP solver implements a bottom-up tabulation followed by an explicit backtracking step to reconstruct the set of selected items. The forward pass alone efficiently computes the optimal objective value, but the backtracking

is needed to determine which items contribute to that value. It also ensures that the output format matches that of the brute-force solver.

As shown in Algorithm 1, the forward pass iteratively builds a *table* of size $(n + 1) \times (W + 1)$. The first row is initialized with zeros. For each cell, the algorithm decides whether to include the current item. If the item's weight exceeds the column's capacity, the value is carried over from the cell above. Otherwise, it takes the maximum of including and excluding the item.

Once the table is filled, the backtracking starts from the bottom-right cell, which holds the optimal value for the problem. To recover the chosen items, the algorithm compares the value in the current cell with the value directly above it. If both are identical, the current item did not contribute to an improved total value and is not part of the solution, so the algorithm moves one row up. Conversely, if the values differ, the item must have been included. Therefore, it is added to the solution set and the algorithm moves diagonally to the row above, reducing the considered capacity by the item's weight. This reverse traversal continues until the top row is reached and the optimal knapsack configuration is reconstructed.

4.2 Implementation of the Slack-based QUBO

`create_linear_terms():`
for $i \leftarrow 0$ **to** $n-1$ **do**
  $linear[(x_{i+1}, x_{i+1})] \leftarrow -v_i + \lambda w_i^2 - 2\lambda w_i W$;
end
for $i \leftarrow 0$ **to** $N-1$ **do**
  $linear[(s_i, s_i)] \leftarrow \lambda 2^{2i} - 2\lambda 2^i W$;
end

`create_quadratic_terms():`
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow i+1$ **to** $n-1$ **do**
    $quadratic[(x_{i+1}, x_{j+1})] \leftarrow 2\lambda w_i w_j$;
  end
end
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** $N-1$ **do**
    $quadratic[(x_{i+1}, s_j)] \leftarrow 2\lambda w_i 2^j$;
  end
end
for $i \leftarrow 0$ **to** $N-1$ **do**
  for $j \leftarrow i+1$ **to** $N-1$ **do**
    $quadratic[(s_i, s_j)] \leftarrow 2\lambda 2^{i+j}$;
  end
end

`set_offset():`
$offset \leftarrow \lambda W^2$;

Algorithm 2: Construction of the slack-based QUBO for the KP, showing how the objective terms, slack variables, quadratic penalties, and offset are assembled from the problem instance.

The implementation of the slack-based quantum approach begins with each problem instance being transformed into a QUBO, as described in Section 2.2.3. A dedicated class takes the problem instance and the penalty parameter λ as input. λ defaults to the baseline $\max_i v_i$. The class first determines the number of slack bits $N = \lfloor \log_2 W \rfloor + 1$, and then assembles the linear and quadratic terms and the offset of the energy function, as shown in Algorithm 2. The item-selection variables are named $x_1, \ldots, x_n$ and the slack variables $s_0, \ldots, s_{N-1}$.

This constructed QUBO is converted into a BQM before sampling. Because the slack-based formulation couples every pair of variables, the resulting

QUBO is fully connected and requires a dense, all-to-all embedding. It is embedded into the Zephyr graph with `minorminer`, a heuristic embedding algorithm from the D-Wave software stack, initialized with a fixed seed for reproducibility.

4.3 Implementation of the Controlled Full Adder QUBO

Before creating the QUBO, each problem instance has to be checked. As introduced in Section 3.1, every problem has to have the form $W = 2^k - 1$ with $k \in \mathbb{N}$. This is what makes the overflow interpretation of the output variables $Z_k, Z_{k+1}, \ldots$ usable, as every variable above position $k-1$ then signals a capacity violation. When the problem instance has to be adjusted, the capacity must be expanded to the nearest valid upper bound $\hat{W} = 2^k - 1$ with $\hat{W} \geq W$. After changing the capacity, the original optimization target has to be restored. The newly created difference between $\hat{W}$ and W has to be permanently filled. Therefore, zero-value dummy items are introduced with a combined weight of $\hat{w} = \hat{W} - W$. It is possible that more than one dummy item has to be generated, since the weight of an item has to follow the structure $w = [1, 2^b - 1]$ as described in Section 3.2.1. This is only necessary for this test dataset and the way the adder is constructed. In general, one item with the necessary weight difference $\hat{W} - W$ would be sufficient. Just adding the dummy items to the problem instance would still result in a different problem. To solve this, each added dummy item has to always be included in the knapsack. In code, this is realized by fixing the dummy items' corresponding decision variables $\hat{x}$ to the one-state. For every other aspect of the adder, the dummy items are treated like any other item in the problem instance.

4.3.1 Creating the QUBO

`create_energy_value(problem):`
linear $\leftarrow \{\}$;
foreach $(i, val) \in$ *`enumerate(problem['values'])`* **do**
  linear$[x_{i+1}] \leftarrow -val$;
end
return *`dimod.BQM`*(*linear*);

Algorithm 3: Creation of the energy value function, assigning the negated item values to the linear terms of the BQM and returning the corresponding model.

For creating the QUBO, the first step is to create the profit function. This is done by negating the profit function of the original problem to represent the highest value as a state of the lowest energy. Each value will be mapped

to the corresponding item-selection variable x_i and added to the linear terms of the QUBO as shown in Algorithm 3.

The second step is to create the problem-specific adder structure as shown in Figure A.1. A simplified overview of the construction process is given in Algorithm A.1. Firstly, for each weight w_i, the bit positions are explicitly checked. As explained in Section 3.1, if the j-th bit of w_i can be set to 1, the corresponding variable $V_{i,j}$ is created and added to the j-th column of the adder structure. This is done until all necessary $V_{i,j}$ variables are included. After this, the first part of the CFA structure is built. As seen in Algorithm A.1, for each column, groups of three variables are reduced with FAs. The outgoing sum variable is appended in the same column, while the outgoing carry variable is forwarded to the next more significant column. This reduction is repeated until less than three variables are left in each column. In the next and last step, the remaining variables of a column are mapped to the knapsack-weight Z_j. This is done either by using direct mapping, a HA, or a FA, respectively, depending on the number of remaining variables. In this way, the outputs $Z_0, Z_1, \ldots$ form the binary representation of the total packed knapsack weight.

The third and last step for creating the QUBO is to represent the capacity constraint of the knapsack. It has to be guaranteed that any infeasible solution will always have a higher energy than any feasible solution with optimal profit. Therefore, a penalty is set to all the Z_j which would represent a knapsack weight, where the constraint is violated. To do this, these variables are identified by checking if the binary weight 2^j of the variable Z_j is larger than the adapted capacity of the knapsack. These are the variables that can only be set to 1, when the calculated knapsack weight exceeds the allowed range. All the overflow variables are passed to a small energy-constraint function that adds a positive linear penalty to each of them. This works in the same way as the profit function in the opposite direction. While the item values are encoded as negative terms in the profit function, the overflow variables are encoded as positive terms. This way, every infeasible solution's energy is pushed clearly above the feasible ones.

4.3.2 Scaling the BQM

One crucial part for the CFA structure is to find a good way to scale the penalty for each adder. To understand why this is necessary, it is important to understand how the adders are implemented internally. An adder, whether it is a HA or a FA, is nothing more than a QUBO where the inputs and outputs are represented by qubits. A penalty is set to all the combinations of inputs and outputs which would represent a wrong calculation for the adder. This penalty is set to 1 by default, which means that the energy of a state where the adder calculates wrong is one higher than the energy of a state where the adder calculates right. The penalty has to be increased so

that infeasible solutions do not become more favorable than feasible ones.

The BQM can be scaled by invoking a method, which multiplies all the coefficients of the BQM with a common factor λ. Finding a good λ is a task in itself. Especially for such a new approach. If the λ is too small, the annealer finds a state where the adders do not calculate the correct total weight of the knapsack. The penalty for a wrong calculation would be smaller than the profit for the wrong calculation. This would lead to false calculations of the annealer, since it would likely optimize by picking every item.

`lambdafunction(weights, j, counter):`
$counter \leftarrow counter + 1$;
$\lambda \leftarrow \max(weights) + 2 \cdot j + counter$;
return λ, *counter*;

Algorithm 4: Heuristic scaling of the adder penalties, showing how λ is derived from the current weight range, bit position, and counter value for each adder instance.

What proved to be a good λ was a linear function shown in Algorithm 4, which is called for each adder after it is created. It starts with $\max(weights)$ so that the penalty still depends on the size of the current instance instead of using a fixed value. The term $2 \cdot j$ makes the penalty larger for adders in higher columns, because higher columns stand for larger binary weight values that are calculated. This alone would still not be enough, because a carry can also affect the next column, so the penalty must not depend on j only. A counter makes the penalty grow from one adder to the next, which prevents later adders from becoming too weak when more adders are added to the structure. In this way, the penalties stay large enough to punish wrong calculations, but do not create unnecessarily large coefficients.

Another penalty that has to be added is the penalty for the overflow variables $Z_k, Z_{k+1}, \ldots$. As the overflow variables only have to rule out overflow states, the chosen λ can be set comparatively high at $\lambda = \sum_{i=1}^{n} v_i$ without causing the same fine-tuning problems as in the adder scaling.

To reassure that the λ values are large enough, a checkroutine is implemented to examine the samplesets for feasibility. It checks whether all samples in a sampleset with energy $\leq$ ground-state correspond to optimal item-selections. If this would return false, it would mean that there is a favorable state where more items are selected than the knapsack can hold. Therefore, the adders did not calculate the correct total weight of the knapsack. The auxiliary variables are ignored as they are only needed for the internal construction of the QUBO. Only the x_i variables are relevant for the final solution, since they represent the item-selection.

4.3.3 Optimization

After creating the CFA QUBO, there is still room to optimize it to increase successful embeddings and improve performance. A QUBO can be optimized by shrinking it, which is done by reducing the number of variables and interactions. One way to do this is to fix variables that are always zero or one (Lewis and Glover, 2017). This has been done for the dummy items. Another way to reduce the number of variables is to remove variables that have a one-to-one relation. This is the case for all $V_{i,j}$, which are directly correlated with the item-selection variable x_i and are only active when the corresponding item is selected. Instead of enforcing this with a coupling, each $V_{i,j}$ is directly substituted by the corresponding x_i, without changing the problem. This conserves both couplings and qubits and reduces the QUBO by an average of $\frac{bn}{2} - n$ variables as there are on average $\frac{bn}{2}$ item weight bits $V_{i,j}$ and n item-selection variables which substitute all $V_{i,j}$. While the overall order of adding the different QUBO components to the BQM is theoretically flexible, it must be carefully managed in practice so that all prerequisite variables are already present before applying operations such as the variable contractions.

5 Results

In this section, the empirical results of the implemented algorithms are presented. The evaluation focuses on documenting the raw performance and resource requirements of each approach across the generated benchmark dataset.

5.1 Classical Approaches

The classical algorithms serve as an exact baseline for the evaluation. To understand their performance characteristics, runtime and memory consumption were measured across all benchmark instances.

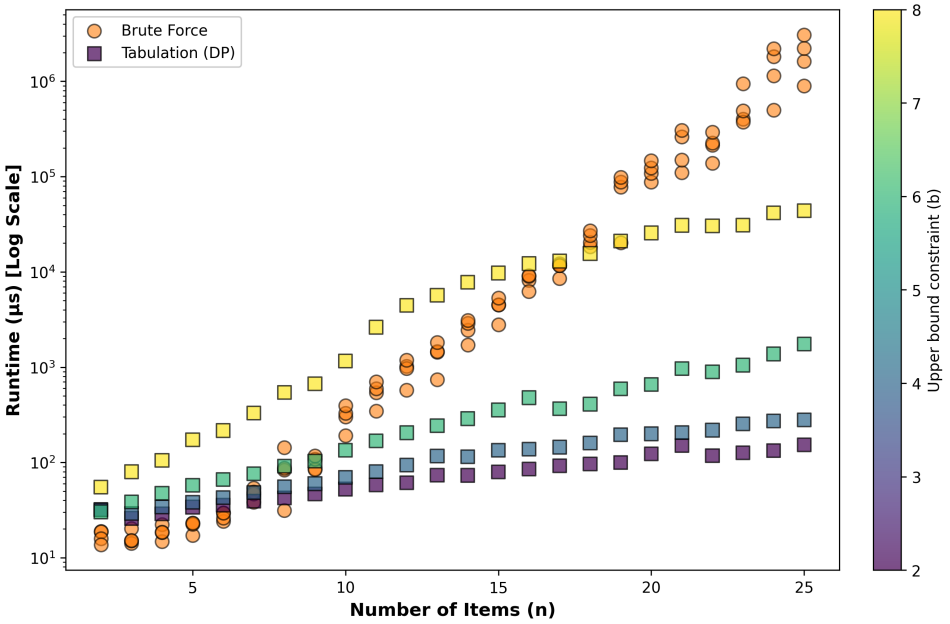

Figure 3: Runtime of the brute-force and DP solvers across the benchmark instances, plotted against the number of items on a logarithmic microsecond scale. The color coding for the DP indicates the corresponding upper bound for values and weights $(2^b - 1)$ of the respective instance.

When observing the runtime behavior in Figure 3, the fundamental algorithmic differences become clear. The brute-force solver shows a strict exponential growth, indicated by the approximately linear upward trend on the logarithmic scale. This behavior is expected, given the algorithm's exhaustive traversal of the binary decision tree. The vertical spread within each item count is driven by the randomized relative capacity constraints $(30\% \pm 10\%)$ applied during data generation. Since the brute-force algorithm prunes branches of its decision tree as soon as the current weight exceeds the

capacity, instances with a lower relative capacity trigger this cutoff earlier. Analyzing each item count group in itself, the instances with the lowest relative capacity in each group are also the ones with the shortest runtime in 22 out of 24 cases. Consequently, exploring fewer branches results in shorter runtimes. In contrast, a higher relative capacity allows more items to be added to the tree before the pruning occurs, which increases the overall runtime. In 22 out of 24 cases, the instance with the highest relative capacity in a group is also the one with the longest runtime. The DP approach, on the other hand, does not form a single smooth curve but rather splits into four distinct bands. These bands correspond to the different capacity ranges defined in the benchmark generation. Since the tabulation runtime is pseudo-polynomial, it depends not only on the number of items but also heavily on the capacity of the knapsack which is determined by the random generation of the item weights and their upper bound. Instances with larger capacities require larger tables which increases their runtime.

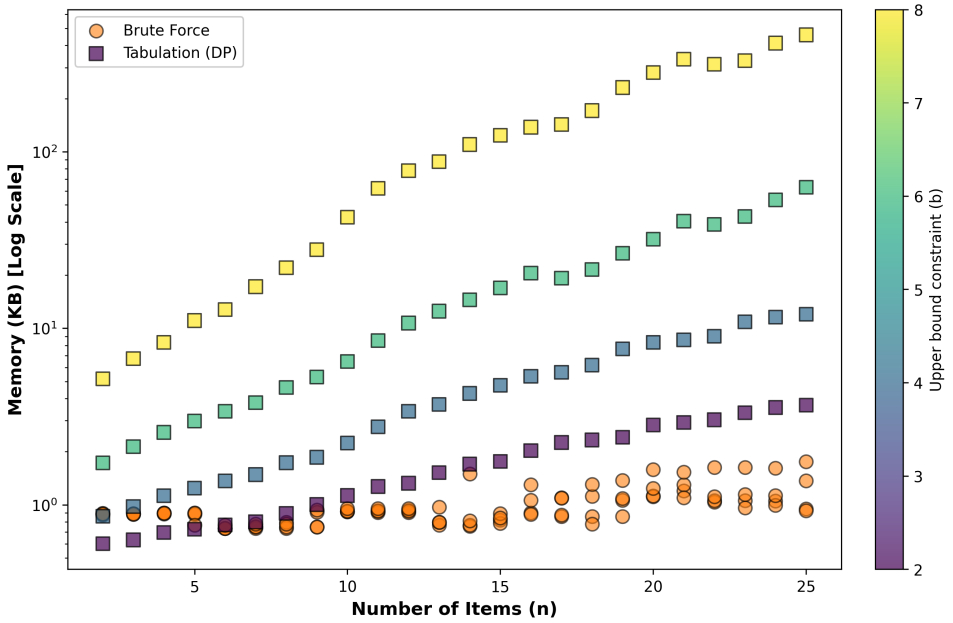

Figure 4: Memory consumption of the brute-force and DP solvers across the benchmark instances, plotted against the number of items on a logarithmic kilobyte scale. The color coding for the DP indicates the corresponding upper bound for values and weights $(2^b - 1)$ of the respective instance.

This runtime advantage of the DP approach comes with a direct trade-off in memory consumption. As shown in Figure 4, the memory usage of the brute-force solver remains in a relatively narrow, low range for most instances. This matches the expectation for a recursive search that only

needs to store the current call stack. In contrast, the tabulation solver's memory usage is primarily driven by the $(n+1) \times (W+1)$ matrix, leading to a significant increase as the problem scales both in terms of items and capacity. Furthermore, it shows a similar banded structure as seen in the runtime plot. Because the memory consumption scales linearly with the knapsack capacity, instances with larger upper bounds for weights allocate proportionally larger matrices. Consequently, the vertical gaps between these bands directly visualize the jumps between the different weight and capacity ranges.

5.2 Slack-based QUBO

Shifting to the quantum-based approaches, the slack-based QUBO provides the initial baseline for problem solving via quantum annealing. The primary metric for the feasibility of this approach is its ability to find an optimal combination of items.

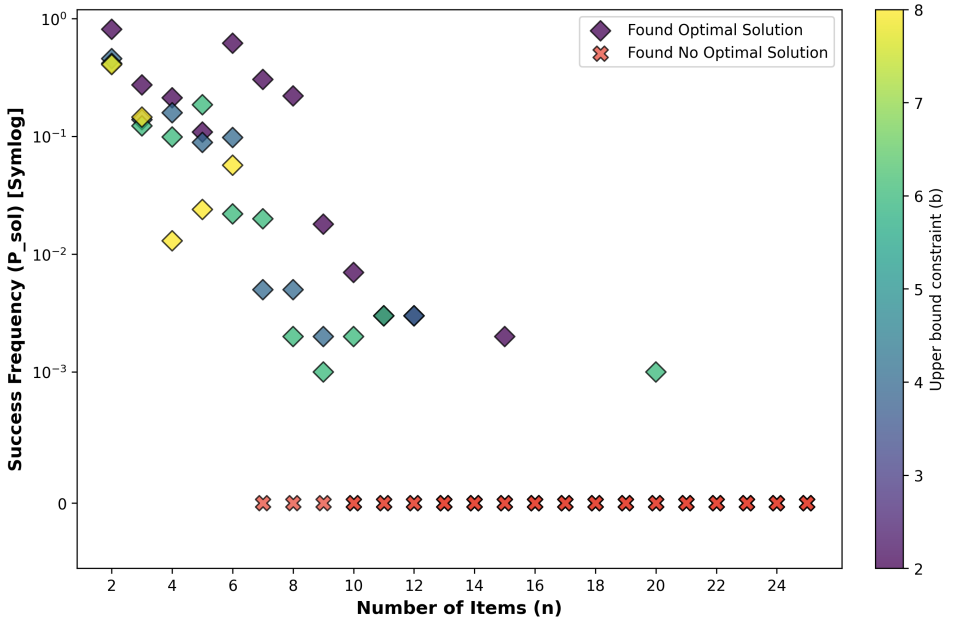

Figure 5: Success frequency of the slack-based QUBO across the benchmark instances on a symmetric logarithmic scale. Diamonds mark instances where at least one optimal solution was found. Red crosses mark instances where no optimal solution was found. The color coding for the optimal instances indicates the corresponding upper bound for values and weights $(2^b - 1)$ of the respective instance.

Figure 5 shows that the encoding achieves high success frequencies for smaller instances. However, the probability of finding an optimal solution declines as the number of items grows. On average, the solver has a success

frequency of 13.7% for finding at least one optimal solution. For larger problem sizes, an increasing number of instances fail to produce an optimal solution entirely. The successful instances remain scattered across the problem size, suggesting that the structure of the individual instance strongly influences the solver's success frequency. Although no distinct band-pattern emerges, the annealer struggles to find optimal solutions as the problem complexity increases both in terms of items and capacity.

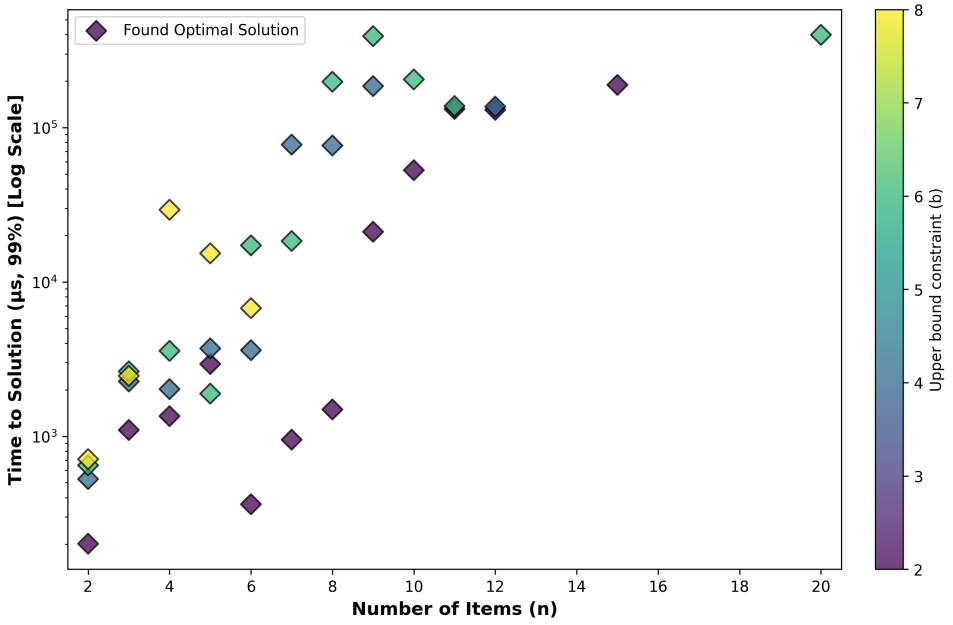

Figure 6: Time to solution at 99% confidence (TTS_{99}) for the slack-based QUBO, plotted against the number of items on a logarithmic scale. Unsolved instances are omitted. The color coding indicates the corresponding upper bound for values and weights $(2^b - 1)$ of the respective instance.

This decline in success probability directly impacts the theoretical time required to guarantee a solution with 99% confidence. Figure 6 reveals that the TTS_{99} generally increases with the problem size. Even on a logarithmic scale, the datapoints rise sharply. The values span several orders of magnitude, which further emphasizes the visible variation between simple and difficult instances. Since the quantum annealer operates with a fixed annealing time per sample, the physical duration of a single read remains essentially constant across all instances. Consequently, the calculation of the TTS_{99} metric is almost exclusively dictated by the success probability. This strict mathematical dependency explains why the sharp upward trend of the TTS_{99} acts as a direct, inverse reflection of the steep decline previously observed in the success frequency Figure 5.

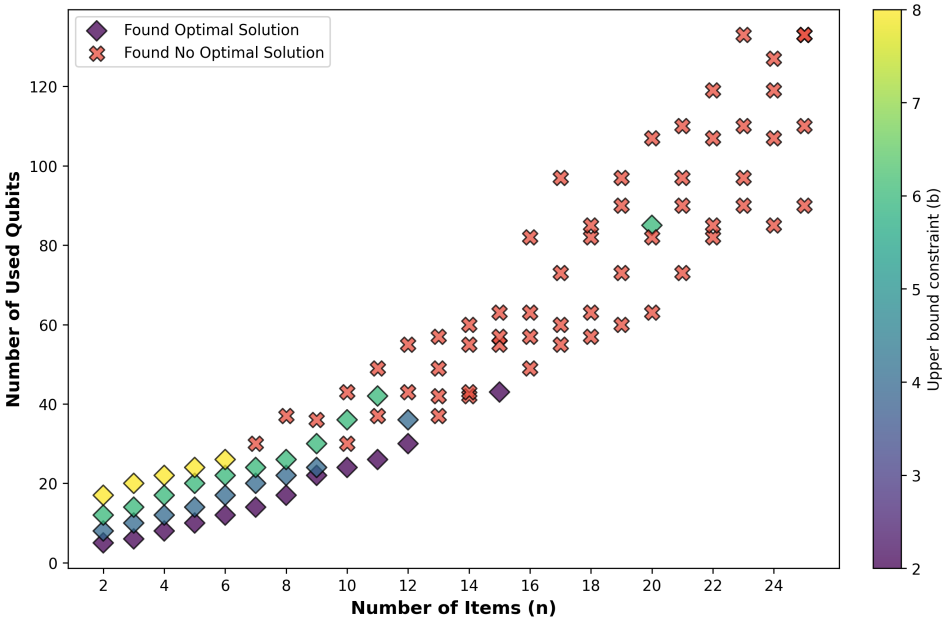

Figure 7: Number of physical qubits required for the slack-based QUBO embedding as a function of the number of items. The color coding indicates the corresponding upper bound for values and weights $(2^b - 1)$ of the respective instance. Solutions where no optimal solution was found are marked with red crosses.

A key factor contributing to this performance degradation is the physical embedding on the QPU. As shown in Figure 7, the number of required physical qubits follows a clear upward trend with a maximum of 133 qubits required for embedding the largest instance. Above a certain threshold of used qubits, the annealer's ability to find optimal solutions decreases drastically. In the lower half of the problem size, the instances can be directly divided into four distinct bands depending on the upper bound of the weights. This illustrates that higher weight limits also demand more qubits to embed the problem. Notably, a single instance at $n = 20$ yielded exactly one optimal solution, despite being surrounded by similar instances for which no optimal solution was found. This behavior can be dismissed as mere chance, given the probabilistic nature of the annealer.

5.3 Controlled Full Adder QUBO

The CFA QUBO represents the alternative arithmetic encoding strategy. Its raw performance is evaluated under the same conditions as the slack-based approach to determine its practical viability.

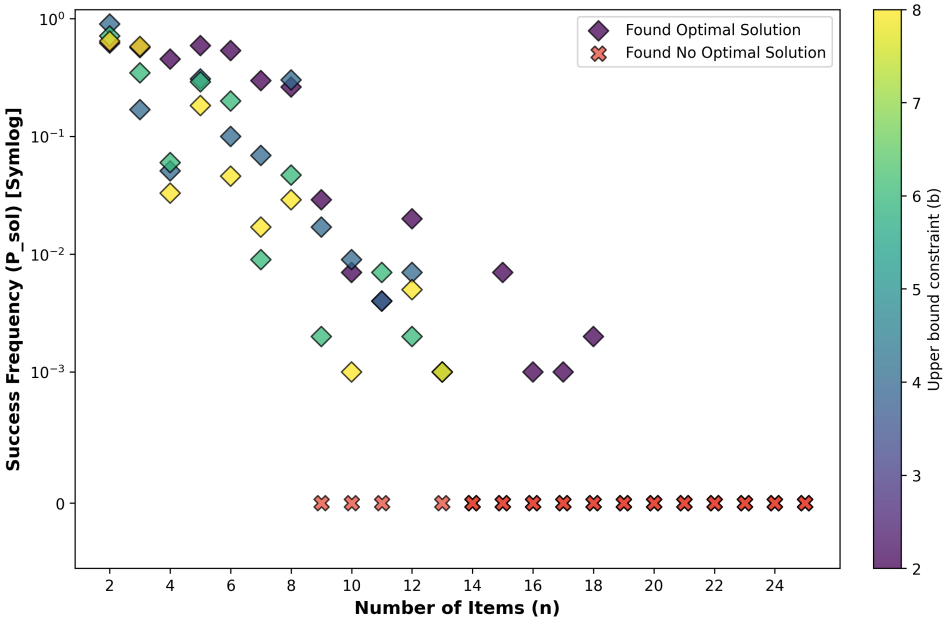

Figure 8: Success frequency of the CFA across the benchmark instances on a symmetric logarithmic scale. Diamonds mark instances where at least one optimal solution was found. Red crosses mark instances where no optimal solution was found. The color coding indicates the corresponding upper bound for values and weights $(2^b - 1)$ of the respective instance.

Figure 8 illustrates the success frequency of the CFA. While a decline in success probability is evidently present as the problem size increases, the degradation is noticeably more gradual compared to the slack-based QUBO. The solver sustains a denser cluster of successful reads across the entire problem size, avoiding a sharp drop-off. The CFA has an average success frequency of 17.8% for finding at least one optimal solution for the given dataset. Instances that fail to yield an optimal solution still appear in the upper half of the problem range. However, a substantial number of larger benchmark instances maintain comparatively high success frequencies. This suggests that the CFA formulation remains considerably more stable under increasing problem complexity. Like the slack-based approach, the success frequencies are scattered across the item range without forming distinct bands.

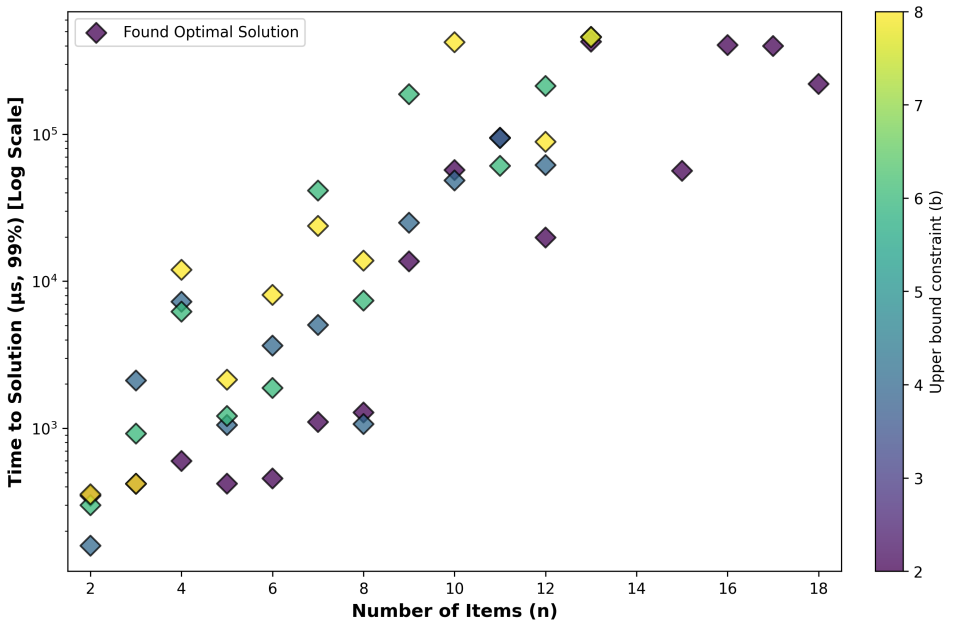

Figure 9: Time to solution at 99% confidence (TTS_{99}) for the CFA QUBO, plotted on a logarithmic scale. Unsolved instances are omitted. The color coding indicates the corresponding upper bound for values and weights $(2^b - 1)$ of the respective instance.

As a direct consequence of these more stable success frequencies, the TTS_{99} metric presented in Figure 9 shows a relatively flatter upward trend. While the broad vertical spread confirms that certain harder instances still demand a considerably higher sampling effort, the CFA formulation effectively lessens the fast growth in execution time for larger instances.

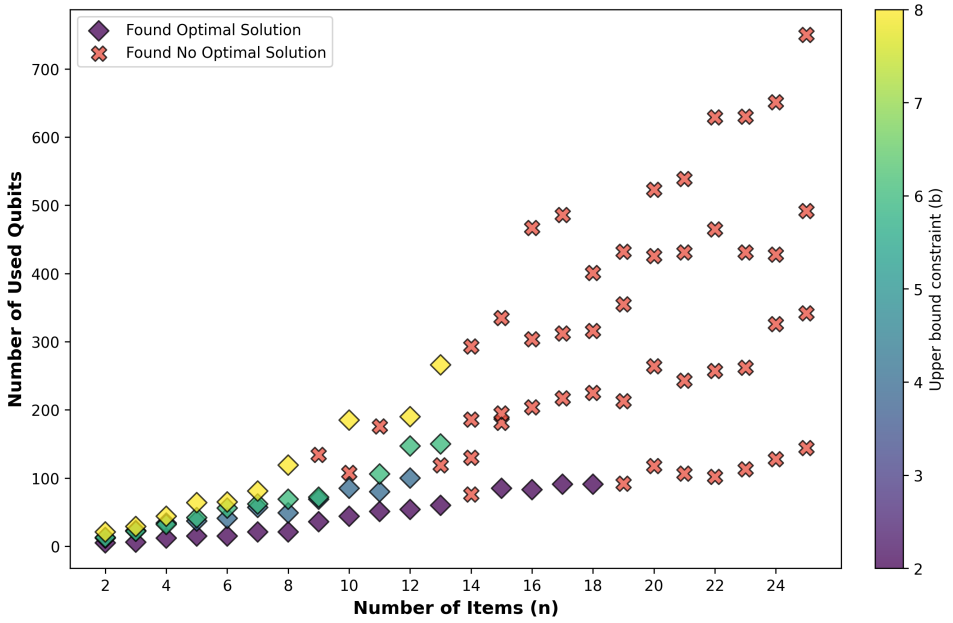

Figure 10: Number of physical qubits required for the CFA QUBO embedding as a function of the number of items. The color coding indicates the corresponding upper bound for values and weights $(2^b - 1)$ of the respective instance. Solutions where no optimal solution was found are marked with red crosses.

The most defining characteristic of the CFA approach is its demanding resource consumption. As shown in Figure 10, qubit usage increases drastically with the number of items. The largest instances require up to 750 qubits. Similar to the slack-based approach, the required qubits distribute into four distinct bands depending on the bit-width b. A larger b shifts an instance into a higher band. Since the item weights are represented in binary, the required number of qubits is solely determined by their bit-width b and the number of items n, not by the absolute magnitude of the weights. Therefore, instances with the same bit-width and number of items would need a similar number of qubits even if their weight values differ. Despite this severe resource consumption, the CFA formulation's arithmetic representation allows it to maintain a much higher success frequency as seen in Figure 8, even as the qubit count vastly grows.

6 Comparative Analysis

While the previous section reported the independent characteristics and metrics of each implemented solver, this chapter provides a direct, comparative evaluation of their relative qualities. By shifting the focus from isolated observations to a systematic analysis, the objective is to uncover the specific trade-offs in each computational paradigm for solving the KP. Firstly, solvers are evaluated within their own paradigms to understand specific algorithmic trade-offs. This is followed by a cross-paradigm comparison to assess broader hardware scaling between all solvers.

6.1 Intra-Paradigmatic Comparison

The analysis begins by comparing the classical brute-force and DP approaches to highlight their specific time-memory trade-off. Subsequently, the quantum slack-based and CFA formulations are evaluated against each other regarding solution quality and resource consumption.

6.1.1 Classical Approaches

Runtime Comparing both classical algorithms, Figure 3 already established that the DP algorithm features a superior runtime scaling compared to the brute-force approach. Although the brute-force solver is faster for smaller instances with large capacities, the DP method consistently outperforms it as the problem size increases. For this dataset, this happens already at around $n = 10$ items for the instances where the weights and values do not exceed $2^6 - 1$. For instances with weights and values up to $2^8 - 1$, the threshold is crossed after around $n = 18$ items. Looking at the growth of both algorithms, it is clear that the DP will eventually outperform the brute-force algorithm for any instance size, when the number of items grow large enough. This matches with the theoretical runtime of both algorithms, which is exponential for the brute-force and pseudo-polynomial for DP. $O(2^n)$ will always grow faster than $O(n \times W)$ when $n \to \infty$. The only case in which the brute-force can consistently outperform the DP is when the capacity W grows exponentially with the number of items n.

Resource Consumption This runtime advantage, however, requires a substantial trade-off in memory allocation. As previously shown in Figure 4, the memory usage of the DP algorithm scales strictly with the instance's capacity. For almost all benchmark instances, with just a few exceptions, the memory consumption drastically exceeds that of the brute-force solver. In extreme cases, the DP requires over 260 times more memory. Since the DP algorithm constructs an $(n+1) \times (W+1)$ matrix to store intermediate results to prevent redundant subproblem calculations, this memory overhead is an

inherent and unavoidable cost for achieving its pseudo-polynomial runtime. The brute-force algorithm, conversely, only needs to maintain the current recursive call stack, keeping its memory footprint consistently minimal.

Solution Quality Since both the brute-force and DP solvers operate as exact, deterministic algorithms, both reliably find a global optimal solution for every feasible problem instance. Consequently, there is no variance in solution quality between the two approaches.

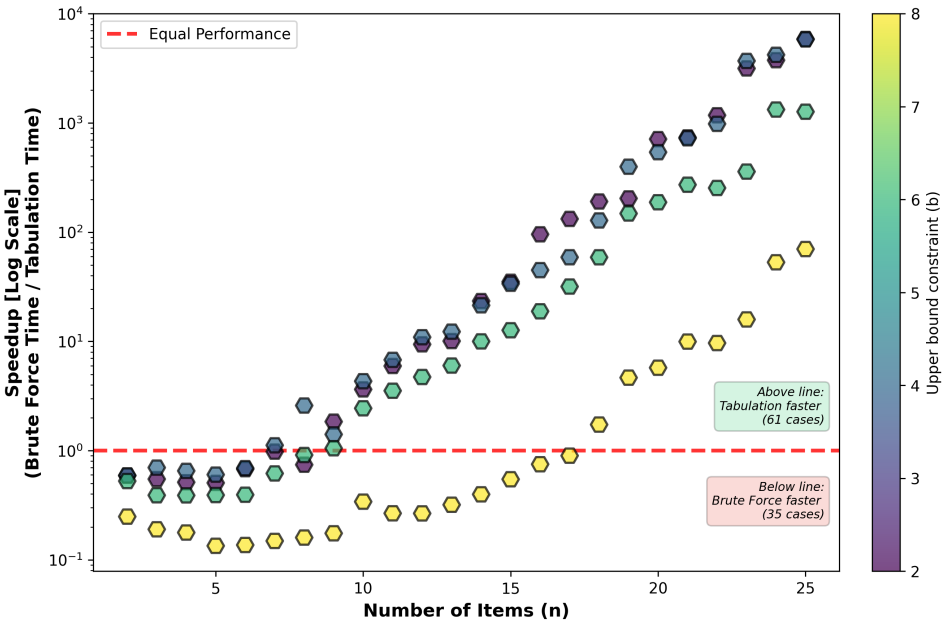

Figure 11: Runtime speedup of the DP solver compared to the brute-force solver across the benchmark instances on a logarithmic scale, plotted against the number of items. The speedup is calculated as the ratio of the brute-force runtime to the DP runtime. A red dashed line indicates a speedup of 1, where both algorithms perform equally.

Scalability The scalability of the classical algorithms is represented in Figure 11, which visualizes the relative runtime speedup. The mainly linear upward trend on the logarithmic scale indicates an exponential growth in the speedup factor as the number of items increases. However, the data also reveals that the DP algorithm is not universally superior. For over one-third of the benchmark instances, specifically those characterized by a low item count combined with a large capacity constraint, the brute-force approach solves faster. In these cases, the heavy initial overhead of allocating and inserting into the DP matrix outweighs the benefits of avoiding

redundant calculations. Therefore, the threshold at which the dynamic programming approach outperforms the brute-force, heavily depends on the specific parameter ratio of n and W, which for the dataset is solely dependent on the upper bound for the weights. As described in the runtime analysis, the DP algorithm guarantees a cross-over point for sufficiently large problem sizes, as long as the capacity does not grow exponentially with the number of items.

6.1.2 Quantum-based Approaches

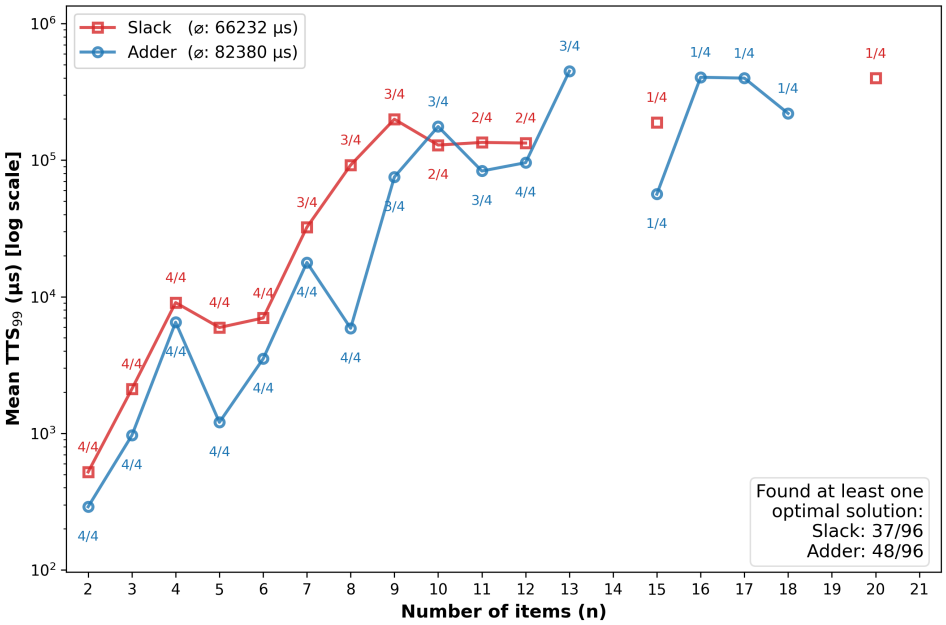

Figure 12: Mean TTS_{99} of the slack-based and CFA QUBO approaches across the benchmark instances on a logarithmic microsecond scale, plotted against the number of items. The fractions for each datapoint indicate the number of instances where at least one optimal solution was found over the total number of instances for that item count.

Runtime Shifting to the quantum paradigm, Figure 12 contrasts the mean TTS_{99} for both QUBO approaches. An initial evaluation of all successful instances reveals that the CFA approach is significantly more capable, finding at least one optimal solution for 48 out of 96 problems with a mean TTS_{99} of $82380\,\mu s$. In comparison, the slack-based approach only found 37 instances, recording a lower mean TTS_{99} of $66232\,\mu s$. Although the slack-based approach appears to have a lower mean TTS_{99}, Figure 12 shows that except for $n = 10$, the CFA consistently outperforms the slack-based approach across all problem sizes.

For $n = 10$, another phenomenon occurs, which explains why the overall mean TTS_{99} for the slack-based approach is lower, although graphically the CFA seems to perform better. At this specific problem size, the slack-based solver only managed to find the optimum for two, seemingly simpler instances than the CFA. The CFA however, successfully solved multiple, seemingly harder instances at $n = 10$. Later in Figure 15 it will be shown that the CFA actually has on average a slightly higher success frequency for $n = 10$. While this should mean that the TTS_{99} for the CFA should be lower than for the slack-based approach, the actual annealing time T_A for a single read, which is also used to calculate the TTS_{99}, is not fixed in practice. When only a small difference exists in the success frequency, the TTS_{99} can be influenced by small variations in the annealing time which happened for $n = 10$ and results in a higher mean TTS_{99} for the CFA.

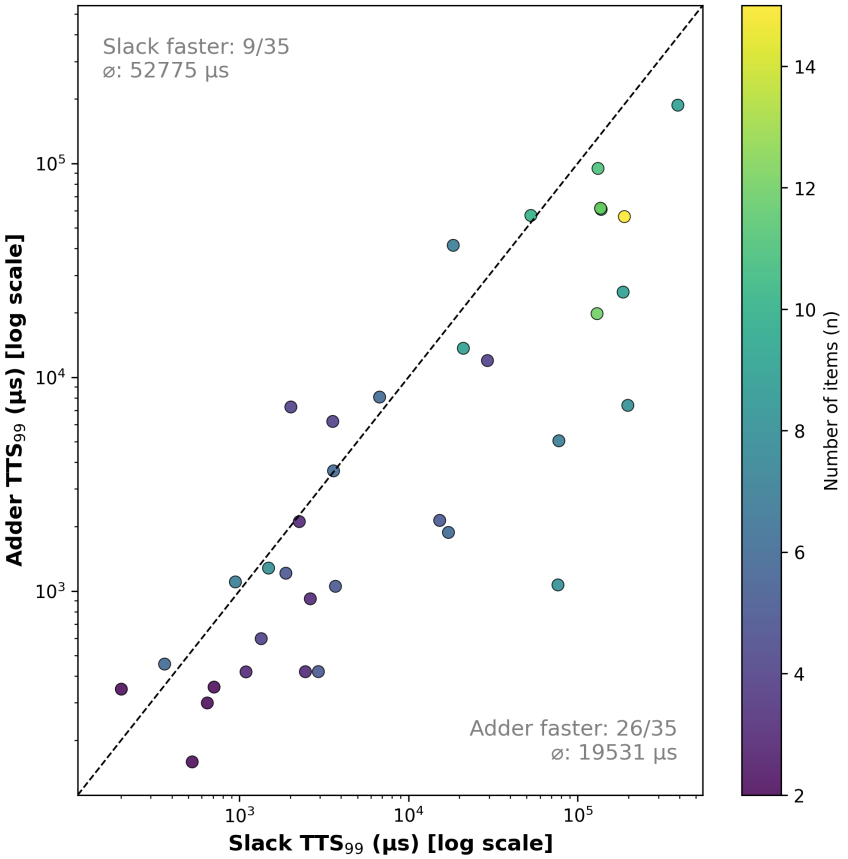

Figure 13: Per-instance comparison of TTS_{99} on the 35 instances solved by both approaches. Each point is one benchmark instance, with the slack-based TTS_{99} on the x-axis and the CFA TTS_{99} on the y-axis, both on a logarithmic microsecond scale. The number of items n is encoded by color, and the dashed diagonal marks equal TTS_{99}.

To eliminate this statistical bias for a mathematically fair comparison, a direct evaluation was done strictly on the intersection of both datasets. Figure 13 shows the 35 benchmark instances successfully solved by both approaches. It demonstrates that across the shared instances, the mean TTS_{99} for the CFA approach drops to 19531 μs, whereas the slack-based approach averages 52775 μs. Evaluated by their means, the CFA formulation consistently solves identical problem instances roughly 2.7 times faster than the slack-based counterpart. The CFA achieves a lower TTS_{99} on 26 of the 35 shared instances, while the slack-based approach is faster on only 9. This confirms that the CFA is faster not only on average but on most individual problems as well. This matches with the overall picture in Figure 12 and Figure 13, where the CFA approach maintains a clear lead across most problem sizes.

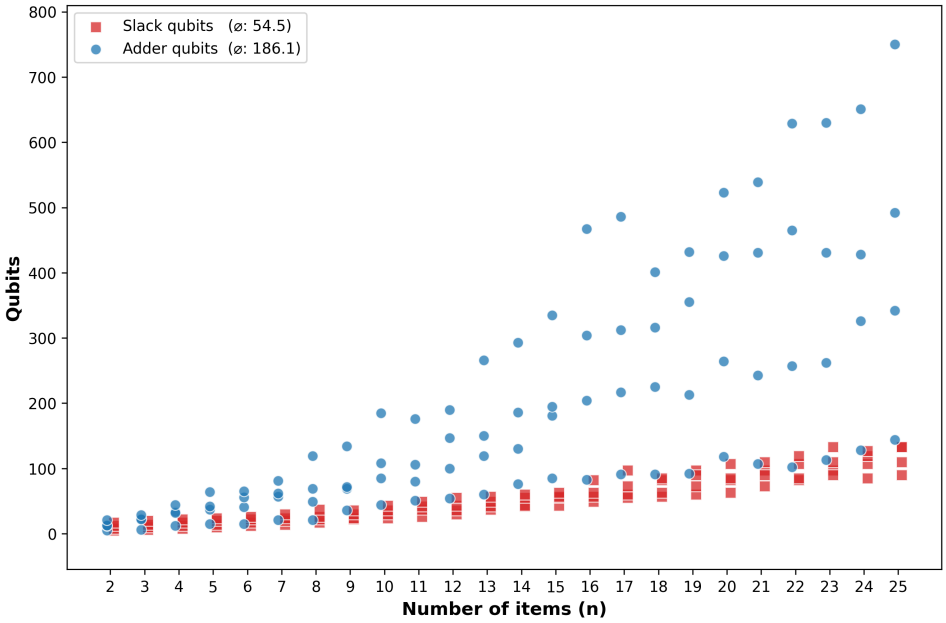

Figure 14: Qubit consumption of the slack-based and CFA approaches across the benchmark instances plotted against the number of items.

Resource Consumption The extended solving capability of the CFA formulation comes at a severe hardware cost. Figure 14 directly compares the required physical qubits for embedding the problems onto the QPU. The data demonstrates that the slack-based formulation is drastically more resource-efficient, utilizing fewer qubits in 94 out of the 96 benchmark instances. The remaining two instances require an equal amount. On average, the slack-based approach requires 54.5 qubits, reaching a maximum of 133. Conversely, the CFA formulation demands an average of 186.1 qubits, peaking at 750

for the most complex instance. Overall, the slack-based approach operates on average on no more than 29% of the qubit budget required by the CFA encoding.

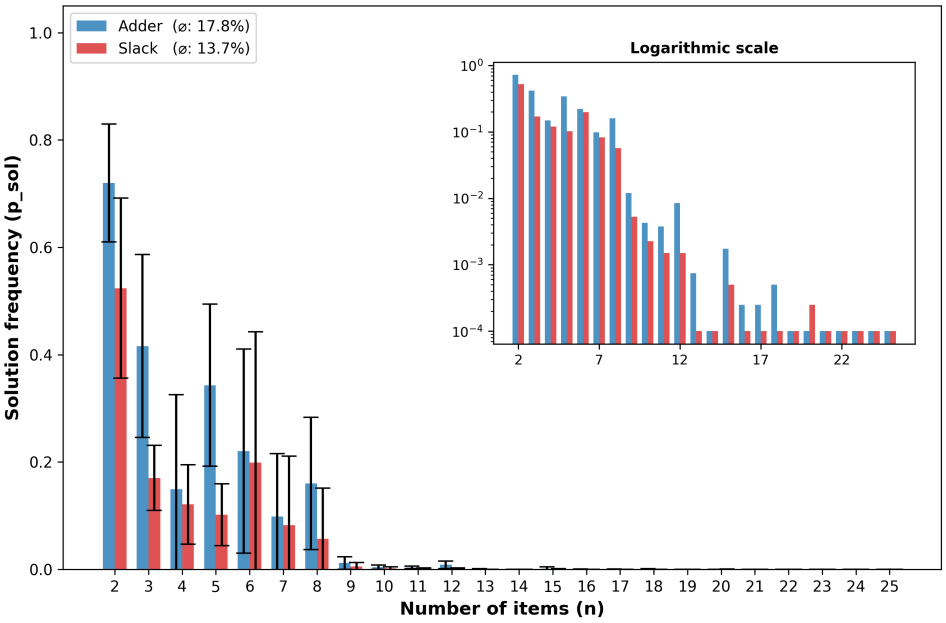

Figure 15: Success frequency (p_{sol}) of the slack-based and CFA QUBO approaches across the benchmark instances, plotted against the number of items. The bars represent the mean success frequency for each problem size, with the standard deviation shown as error bars. The subplot on the right shows the same data on a logarithmic scale to better visualize the differences at lower frequencies. The frequency represents the proportion of sampling runs that find an optimal solution.

Solution Quality Despite its intensive resource demands, the CFA formulation delivers a noticeably superior solution quality. Figure 15 illustrates the success frequencies (p_{sol}) of both approaches. Across the entire dataset, the CFA approach yields a higher average success frequency of 17.8% for finding at least one optimal solution, compared to the slack-based average of 13.7%. This indicates that the arithmetic structure of the CFA encoding generates a more favorable energy landscape for the quantum annealer, allowing it to more effectively find optimal solutions. To ensure a fair comparison, the success frequencies were also evaluated strictly on the intersection of the successfully solved instances. For these 35 shared instances, the CFA approach achieves an average success frequency of 24.2%, while the slack-based approach averages 14.5%. This means that on identical problem instances,

the CFA formulation is approximately 1.7 times more likely to find an optimal solution in a single sampling run than the slack-based approach.

Although the CFA seems to be very promising in terms of solution quality, it is very important to also look into the deviations in Figure 15. The error bars are large, especially for small problem sizes, and the standard deviations of the two approaches overlap for most item counts. This follows from the small sample of only four instances per problem size, so the success frequency varies strongly between individual instances. Therefore, Figure 12 and Figure 13 should be read as an indication rather than a statistically firm result. The overall averages and the intersection comparison nonetheless point consistently towards the CFA.

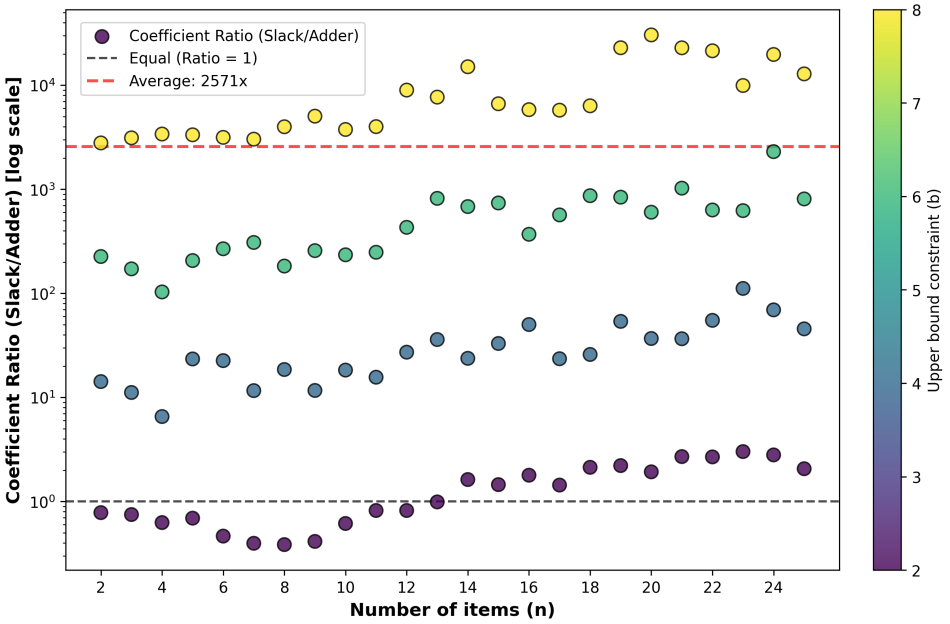

Figure 16: Ratio of the maximum QUBO coefficients between the slack-based and CFA approaches across the benchmark instances on a logarithmic scale, plotted against the number of items. The black horizontal dashed line marks a ratio of 1, where both approaches have equal maximum coefficients. Values above this line indicate instances where the slack-based formulation has higher maximum coefficients, while values below indicate instances where the CFA formulation has higher maximum coefficients. The red dashed line indicates the average ratio across all instances.

Scalability Finally, analyzing the maximum QUBO coefficients provides crucial insight into hardware scalability and the formulations' sensitivity to rescaling. Figure 14 already established that the CFA formulation demands

a significantly higher qubit count, which is a direct consequence of its more complex arithmetic encoding. Although the qubit count is a critical factor, the maximum QUBO coefficients also play an essential role in determining the practical performance of the annealer. Figure 16 visualizes the ratio of the maximum coefficients between the slack-based and CFA formulations. The CFA approach proves to be significantly more resilient, generating lower maximum coefficients in 84 out of the 96 instances. The slack-based approach only achieves lower coefficients in 12 instances, especially where the item count is small and the weights are limited by $b = 2$. Since current quantum annealing hardware must map these theoretical coefficients to physical analog parameters, rescaling of the QUBO is usually unavoidable. As the largest absolute coefficient in a QUBO is used to scale down the whole QUBO, it is highly preferable to have low coefficients. The red dashed line in Figure 16 indicates the average ratio across all instances, which sits at approximately $2571\times$. This means that on average, the maximum coefficient of the slack-based formulation is around $2571\times$ higher than that of the CFA formulation. This extreme difference in coefficient magnitude suggests that the slack-based approach is heavily sensitive to rescaling. This can lead to a significant loss in solution quality due to the limited precision of the hardware as described in Section 3.1. The plot also clearly shows that the bigger the weights are, the greater the coefficients will be for the slack-based approach. Therefore, the CFA effectively trades an increased physical qubit requirement for crucial analog robustness, allowing it to scale more successfully to instances with large weights as seen in Figure 10. This suggests that there is a trade-off in terms of qubits and coefficients.

6.2 Inter-Paradigmatic Comparison

Having compared the internal trade-offs of the classical and quantum approaches isolated, this section broadens the scope to a cross-paradigm comparison of all approaches. Deterministic CPU-based algorithms and probabilistic quantum annealing represent completely different computational paradigms. Nevertheless, despite their fundamental and hardware-specific differences, they are evaluated against each other across the shared dimensions of time, space, and scalability. Because the two paradigms differ so fundamentally, this comparison can only provide an indication rather than a strict one-to-one comparison of performance. It should be understood as an attempt to relate approaches that are not directly comparable.

6.2.1 Runtime

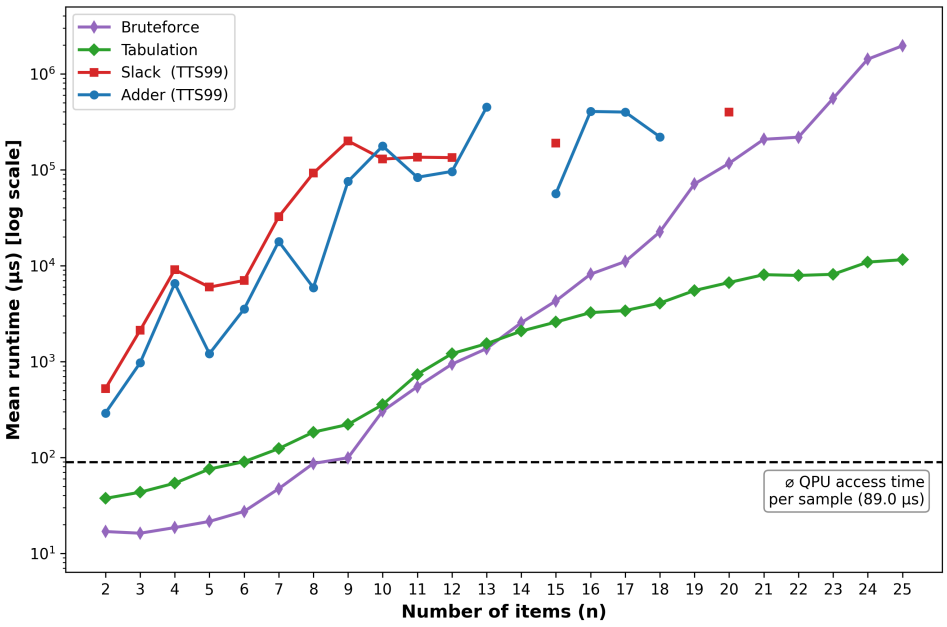

Figure 17: Mean runtime of the classical and quantum-based solvers across the benchmark instances on a logarithmic microsecond scale, plotted against the number of items. Classical performance is measured as direct execution time, whereas quantum performance is represented by the TTS_{99} metric. The dashed line marks the mean QPU access time per sample, comprising the annealing, readout and delay time.

Figure 17 contrasts the measured execution times of the classical solvers with the TTS_{99} of the quantum approaches. These two metrics are not directly comparable. The classical runtime is the time to deterministically compute the exact optimum, while the TTS_{99} is the expected time to observe an optimal solution at least once with 99% confidence. While classical solvers execute a deterministic set of instructions as fast as the CPU clock allows, the quantum TTS_{99} incorporates the probabilistic nature of the annealer. It accounts for the physical duration of the annealing cycles, the readout overhead, and the statistical necessity of drawing many samples to reach a 99% confidence of finding an optimal solution.

It is important to note that the annealer itself is fast. The dashed line marks the mean QPU access time of a single sample at $89\,\mu s$, which already covers the annealing, readout and delay time. An annealing run for a single sample is therefore of a similar order to the classical solvers on small instances. Although the annealing time can have some deviations, it remains roughly constant. The large TTS_{99} values do not stem from a slow annealer but from its limited accuracy. Because the success probability p_{sol} is low, many

samples are required to reach the chosen confidence.

The confidence level of 99% in the TTS_{99} is itself an arbitrary choice. A different threshold would shift the curves for the quantum approaches accordingly. Therefore, their absolute position should be read as a consequence of this choice rather than as a fixed hardware speed.

Taken together, the classical solvers are faster than the quantum approaches across all evaluated problem sizes according to the TTS_{99} metric. This reflects the current accuracy of the hardware rather than a speed limit, as the annealer produces individual samples rather quickly. However, it needs many samples to compensate for its low success probability.

6.2.2 Resource

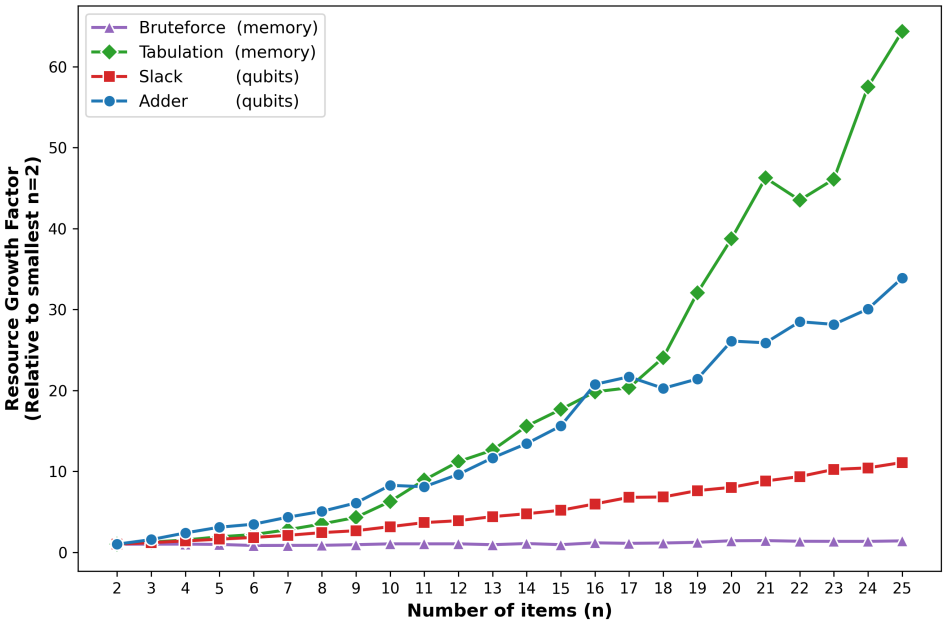

Figure 18: Resource consumption growth factor of the classical and quantum-based solvers, plotted against the number of items. Each curve is normalized to its own smallest instance $n = 2$. The figure shows relative scaling rather than absolute consumption. The underlying magnitude is the peak memory in kilobytes for the classical solvers and the number of required qubits for the quantum approaches.

While runtime evaluates the temporal dimension, Figure 18 visualizes the relative physical resource requirements. Although classical kilobytes and physical qubits are not convertible, they both define the absolute limits of problem solvability for their respective architectures. For this reason each approach is normalized to its own smallest instance, so the figure compares scaling trends rather than absolute amounts.

The growth factors differ strongly between the approaches. The brute-force memory stays almost flat, since it only needs to hold the current recursive call stack. The remaining three algorithms grow with the problem size, with the slack-based qubit count growing moderately up until about a factor of 11× for $n = 25$. The CFA qubit count grows more steeply to about 34×, and the DP memory consumption grows the fastest, reaching 64× more memory consumed compared to its smallest-instance at $n = 25$. While the actual memory consumption shown in Figure 4 is effectively negligible on modern classical hardware for this dataset, it represents a strict hardware requirement. In contrast, the quantum paradigms are strictly bottlenecked by the topological availability of physical qubits on the QPU. Figure 14 highlights how quickly the problem formulations, especially for the CFA, consume the available quantum hardware budget for the D-Wave Advantage2_system2.

This cross-comparison of both paradigms reveals a general trade-off. The more time-efficient approaches in each paradigm (DP and the CFA considering the TTS_{99}) require significantly more resources. Furthermore, as the problem size increases, this resource consumption grows drastically. While this comparison is far from perfect, it gives a clear indication of the fundamental resource allocation and scalability differences between the paradigms.

6.2.3 Solution-Quality

For both paradigms, the main difference lies in their deterministic and probabilistic nature. The classical solvers are deterministic solvers. These guarantee finding the optimal solution for every single instance. Their solution quality is therefore consistently optimal. In contrast, the quantum annealer as a heuristic, probabilistic solver is not guaranteed to find an optimal solution in a single run. Due to these fundamental differences, a direct comparison of solution quality is not meaningful. However, the success frequencies of the quantum approaches provide insight into their practical performance.

6.2.4 Scalability

Combining the runtime, resource, and quality metrics reveals the current state and the fundamental bottlenecks of cross-paradigm scalability. As visualized in Figure 18, the memory consumption of the DP approach shows an almost exponential growth. The resource growth for the CFA approach is not as drastic, but it is still quite high relative to the slack-based approach.

This highlights the most critical difference in scalability between the paradigms. Unlike classical RAM, which is rather cheap and easy to expand, every additionally required qubit is a limited asset. There is no such thing as

"just adding more qubits" to the annealer. For now, the physical qubits on the QPU are fixed and finite, and their connectivity is also technically limited by the hardware architecture. Hence, this hardware limitation inflicts a severe physical boundary on the maximum problem size that can be embedded on the annealer, long before classical memory limits are even approached.

Ultimately, this analysis reveals a universal principle across all evaluated solvers. Every implemented approach is fundamentally constrained by the central limits of its own design. While certain algorithms demonstrate superior scalability in specific metrics, they simply shift the computational bottleneck rather than resolving it entirely. In the classical area, the brute-force algorithm is strictly bound by its exponential time complexity. The DP approach, on the other hand, avoids this time-based constraint only to be severely bottlenecked by its massive memory consumption. A notably symmetric trade-off exists within the quantum formulations. The slack-based encoding requires fewer qubits but is heavily restricted by its growing coefficient magnitudes. The CFA formulation reduces this analog sensitivity at the cost of rising physical qubit usage. Consequently, no single approach offers uncompromised scalability. Instead, each solver simply determines which critical resource is exhausted first.

7 Discussion

The previous chapters reported the raw performance characteristics of the four implemented solvers and contrasted them both within and across computational paradigms. This chapter interprets these empirical findings in light of the hypotheses formulated in Section 1.1. It also considers what the observed trade-offs imply for the practical use of quantum annealing in solving the KP. Specific attention is given to the hardware-related and methodological constraints that shaped the experimental outcomes. Understanding these constraints is essential for assessing to what extent the present findings generalize beyond the specific benchmark dataset and QPU generation considered here.

7.1 Hypothesis Validation

H1 (classical baseline). The empirical comparison of the classical solvers in Section 6.1.1 confirms H1 in both parts. As expected, the DP consistently outperforms the brute-force solver in runtime once the instance size exceeds a certain threshold. For the present dataset, this threshold lies around $n = 10$ items for instances with $b \leq 6$, and is shifted to roughly $n = 18$ for the largest value and weight range $b = 8$. This change also confirms the second part of H1. Because the runtime of the tabulation approach is pseudo-polynomial in W, larger capacities shift the point at which its advantage over the exponential brute-force algorithm arises. The data also reveal an unanticipated outcome where roughly one third of the instances, with low item counts and comparatively large capacities, are directly outperformed by the brute-force approach, as seen in Figure 11. This finding entails a crucial extension of H1. The crossover point between the two algorithms is not a fixed instance size, but heavily depends on the ratio between n and W.

H2 (quantum baseline). The results for the slack-based QUBO from Sections 5.2 and 6.1.2 are consistent with H2. The encoding produced a valid, sampleable formulation for every benchmark instance. It found at least one optimal solution for 37 of the 96 instances, with success frequencies and the related TTS_{99} decreasing as n increases (Figures 5 and 6). However, the qualifier "often" in H2 requires closer examination. Even for small instances, the mean success frequency rarely exceeds 0.2 with the exception of all instances of $n = 2$, where the success frequency is 0.4–0.7 (Figure 15). Therefore, multiple samples are usually needed to find an optimal solution. H2 is therefore confirmed, yet "often producing optimal solutions" should be read probabilistically rather than as near-deterministic, even for the smallest instances.

H3 (central hypothesis). The comparison between the slack-based and CFA formulations in Section 6.1.2 provides clear evidence for every dimension of H3. The CFA encoding found at least one optimal solution for

48 of the 96 instances, compared with 37 for the slack-based approach. It achieved a higher average success frequency across the full dataset (17.8% versus 13.7%). This gap is even greater on the 35 instances solved by both approaches (24.2% versus 14.5%, i.e. roughly 1.7× more likely to find an optimal sample in a single run). On these shared instances, the CFA also reached a lower mean TTS_{99} by a factor of approximately 2.7, confirming the improvement in both solution quality and time-to-solution predicted by H3. This advantage, however, is not free. The CFA formulation requires substantially more physical qubits, averaging 186.1 against 54.5 for the slack-based approach (a factor of roughly 3.4). The necessary qubit count for the CFA peaks at 750 qubits for the largest instances. This is precisely the increase in resource demand and embedding complexity anticipated by H3. Given the data, it suggests that the improvement in solution quality is not a simple consequence of using more qubits. The improvement rather comes from the structural properties of the arithmetic encoding itself. The CFA produces lower maximum QUBO coefficients than the slack-based formulation in 84 of the 96 instances, with an average ratio of roughly 2571× (Figure 16). By decoupling coefficient growth from the numerical magnitude of item weights and capacities, the CFA formulation appears to create a more favorable energy landscape for the annealer. One in which feasible and optimal configurations remain distinguishable even after the rescaling to the hardware's limited coefficient range. H3 is therefore confirmed.

7.2 Practical Implications

On current annealing hardware, the linear and quadratic coefficients of a QUBO must fit into a fixed range (Section 3.1). Any QUBO whose coefficients exceed this range must be rescaled before embedding. This rescaling degrades solution quality independently of the encoding's logical correctness. The fact that the CFA produces lower maximum coefficients in the large majority of instances, despite requiring more qubits, shows that algorithmic design choices can directly compensate for this hardware limitation.

A second implication addresses embedding and connectivity which is a quantum annealing aspect that has no counterpart in classical computation. Because the Zephyr graph of the QPU is not fully connected, every logical variable must be mapped onto one or more physically coupled qubits. The quality of this mapping directly affects the solution quality (Section 2.1.2). The CFA's dependency on adder circuits is structurally favorable, since adders only require local connections between neighboring bit positions rather than the dense, all-to-all connectivity required by the slack-based formulation. This local structure is most likely the reason for the smoother decline in success frequency observed for the CFA in Figure 8. However, this advantage is not unconditional. As instances grow, the adder circuits do as well, resulting in chains that could eventually limit the CFA in the same way,

just at a larger problem size than for the slack-based formulation.

Beyond these formulation-specific trade-offs, the results also point to a more general property of quantum annealers. As discussed in Section 6.2.1, the TTS_{99} differences observed in Figure 17 are driven almost entirely by the success frequency p_{sol}. This is due to the physical duration of a single annealing cycle where the annealing time T_A remains approximately constant regardless of the problem's logical size. This points to a fundamentally different cost structure than classical computation. A large portion of an annealer's operating cost does not come from the computation itself, but from maintaining the QPU at operating temperatures. The QPU is constantly cooled to a few millikelvin above absolute zero to be able to create the necessary quantum states. Once this cryogenic environment is created, each individual annealing run consumes almost no additional energy. Quantum annealers are nonetheless already in active use, mainly for research purposes. One reason for their relevance comes not from outperforming classical solvers, but from cost and operating principles that differ fundamentally from those of conventional hardware.

As quantum hardware continues to evolve, improvements in overall hardware performance, including qubit coherence, connectivity, noise reduction, and error rates, may lead to increases in success frequency. Beyond resource availability, the quantum approaches examined in this thesis are bottlenecked by hardware limitations that are actively being addressed through research. More advanced QPU implementations could offer greater coupler density and a larger number of qubits per graph. Until such topological improvements are realized, formulations like the CFA approach show how algorithmic design can reduce these analog hardware limitations. While quantum annealing cannot currently compete with classical solvers for the exact solution of the Knapsack Problem, resource-heavy but coefficient-efficient formulations like the CFA lay important foundational work for when quantum hardware develops in terms of qubit availability and connectivity.

8 Conclusion

This thesis compared classical and quantum-based approaches for solving the 0/1 Knapsack Problem, with the CFA formulation as its central focus. The results show that every approach comes with its own characteristic trade-offs. The brute-force algorithm is simple and memory-efficient, but its exponential runtime growth limits it to small instances. The DP achieves a substantial runtime advantage over brute-force, at the cost of significantly higher memory consumption. Among the quantum-based approaches, the slack-based formulation is comparatively qubit-efficient but suffers from large coefficients. The CFA formulation on the other hand achieves better solution quality at the price of a significantly higher qubit demand. In both paradigms, there is thus no universally best solver. Which solver suits best heavily depends on both the problem instance as well as the available computational resources.

The central finding of this thesis concerns the CFA formulation. Its ability to sustain a higher success frequency despite its substantial resource demands suggests that it models the structure of the Knapsack Problem more effectively than the slack-based encoding. This results in a more favorable energy landscape for the quantum annealer. At the same time, its high qubit demand remains a practical limitation for scaling to larger instances. The Outlook below discusses several potential directions for future research of the CFA formulation.

Beyond these concrete results, the comparison also shows a more general point. For the problem sizes considered here, the most basic classical algorithms are overall the more practical choice. They are exact and therefore always return the optimal solution, and by the TTS_{99} metric they are also faster, whereas the quantum approaches reach the optimum only with a limited frequency. The resource comparison is less direct, since classical memory and physical qubits are not convertible, with memory being abundant on modern hardware while qubits and their connectivity remain the bottleneck for quantum annealing. It therefore seems unlikely that quantum annealers will play an economically relevant role in solving problems such as the KP in the near future. However, observing how this technology develops further remains worthwhile. Classical computing is increasingly approaching physical limits. One well-known example is the miniaturization of transistors, which is progressively reaching its physical boundaries. Against this background, it is crucial that fundamentally different computational paradigms are further researched and refined. Working with a quantum annealer also makes clear that such a paradigm introduces fundamentally new challenges. Embedding a problem onto the hardware graph, for example, is a consideration that simply does not exist for classical computers. Not all of these new properties are disadvantages, however. The annealer takes approximately the same amount of time for each sample, regardless of the problem size. Classical

computing offers no equivalent to this kind of constant-cost sampling.

Overall, the CFA approach proves to be a new and promising way of encoding the KP into a QUBO. It is certainly not the final or perfect solution to encode this problem, yet it represents a solid and valuable alternative that opens up several directions for future research.

8.1 Outlook

The following directions represent possible extensions that naturally arise from the work presented in this thesis, but could not be fully pursued within the scope of a Bachelor's thesis.

Further Optimization While the current CFA implementation already produces a valid QUBO encoding of the KP, enforcing the capacity constraint so that its optimal solutions correspond to optimal item selections, the constructed QUBO leaves substantial room for future optimizations. One of the most critical aspects is the refinement of the penalty scaling factor λ. Properly choosing this factor is crucial to prevent the capacity penalty from either overshadowing the objective function or failing to suppress infeasible states. Thoroughly tuning this parameter could lead to a significant increase in the quality of the solutions.

Furthermore, the results could be improved by using a multi-embedding strategy. Using different physical embeddings for the same problem instance reduces the risk of getting a bad embedding. This usually leads to better success rates. However, at present, this is not rewarding because of the `chain_strength` scaling issues already introduced in Section 3.2.1, which for the CFA results in a degradation of success rates. Fixing this could make parallel embeddings much more useful for the CFA. At the time of writing, the issue is still under investigation by D-Wave.

Broader Benchmark Coverage The benchmark dataset in this thesis was intentionally generated with capacities fixed at approximately $30\% \pm 10\%$ of the total item weight. This choice kept the capacity-to-weight ratio consistent across all benchmark instances, allowing the four solvers to be compared on a controlled and uniform basis rather than introducing an additional varying parameter. Future work could explore whether the observed trends, such as the relatively smooth degradation of the CFA's success frequency, also hold for other capacities.

References

J. I. Adame and P. L. McMahon (2018). *Inhomogeneous driving in quantum annealers can result in orders-of-magnitude improvements in performance*. URL: https://doi.org/10.48550/arXiv.1806.11091 (visited on 03/25/2026).

D. Aharonov et al. (2005). *Adiabatic Quantum Computation is Equivalent to Standard Quantum Computation*. URL: https://doi.org/10.48550/arXiv.quant-ph/0405098 (visited on 03/22/2026).

R. Bellman (1957). *Dynamic Programming*. Princeton University Press.

T. H. Cormen et al. (2022). *Introduction to Algorithms*. The MIT Press.

D-Wave Quantum Inc. (2026a). *D-Wave Coefficient Range Documentation*. URL: https://docs.dwavequantum.com/en/latest/quantum_research/solver_properties_all.html (visited on 05/05/2026).

— (2026b). *D-Wave System Solver Properties Documentation*. URL: https://docs.dwavequantum.com/en/latest/quantum_research/solver_properties_specific.html#advantage2-system2-1 (visited on 03/22/2026).

— (2026c). *D-Wave Zephyr Topology Documentation*. URL: https://docs.dwavequantum.com/en/latest/quantum_research/topologies.html#topology-intro-zephyr (visited on 03/22/2026).

M. R. Garey and D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

T. Imoto et al. (2025). *Universal quantum computation using quantum annealing with the transverse-field Ising Hamiltonian*. URL: https://doi.org/10.48550/arXiv.2402.19114 (visited on 03/22/2026).

R. M. Karp (1972). *Reducibility Among Combinatorial Problems*. Springer.

H. Kellerer, U. Pferschy, and D. Pisinger (2004). *Knapsack Problems*. Springer. URL: https://doi.org/10.1007/978-3-540-24777-7 (visited on 03/21/2026).

M. Lewis and F. Glover (2017). "Quadratic Unconstrained Binary Optimization Problem Preprocessing: Theory and Empirical Analysis". In: URL: https://doi.org/10.1002/net.21751 (visited on 03/21/2026).

A. Lucas (2014). "Ising formulations of many NP problems". In: URL: https://doi.org/10.3389/fphy.2014.00005 (visited on 03/21/2026).

V. Mehta et al. (2022). "Quantum annealing for hard 2-SAT problems : Distribution and scaling of minimum energy gap and success probability". In: URL: https://doi.org/10.1103/physreva.105.062406 (visited on 03/21/2026).

J. A. Montañez-Barrera et al. (2023). *Improving Performance in Combinatorial Optimization Problems with Inequality Constraints: An Evaluation of the Unbalanced Penalization Method on D-Wave Advantage*. URL: https://doi.org/10.1109/QCE57702.2023.00067 (visited on 03/21/2026).

J. Preskill (2018). “Quantum Computing in the NISQ era and beyond”. In: URL: https://doi.org/10.22331/q-2018-08-06-79 (visited on 03/25/2026).

A. P. Prunnen (2022). *The Quadratic Unconstrained Binary Optimization Problem*. Springer. URL: https://doi.org/10.1007/978-3-031-04520-2 (visited on 03/21/2026).

R. A. Quintero and L. F. Zuluaga (2021). *Characterizing and Benchmarking QUBO Reformulations of the Knapsack Problem*. (Visited on 04/09/2026).

C. Sinz (2005). “Towards an Optimal CNF Encoding of Boolean Cardinality Constraints”. In: Springer. URL: https://doi.org/10.1007/11564751_73 (visited on 03/21/2026).

D. Willsch et al. (2025). *The State of Factoring on Quantum Computers*. URL: https://doi.org/10.48550/arXiv.2410.14397 (visited on 03/21/2026).

A Appendix

The complete implementation is available at `https://github.com/MvnSwr/Bachelor-Thesis/releases/tag/v1.0`.

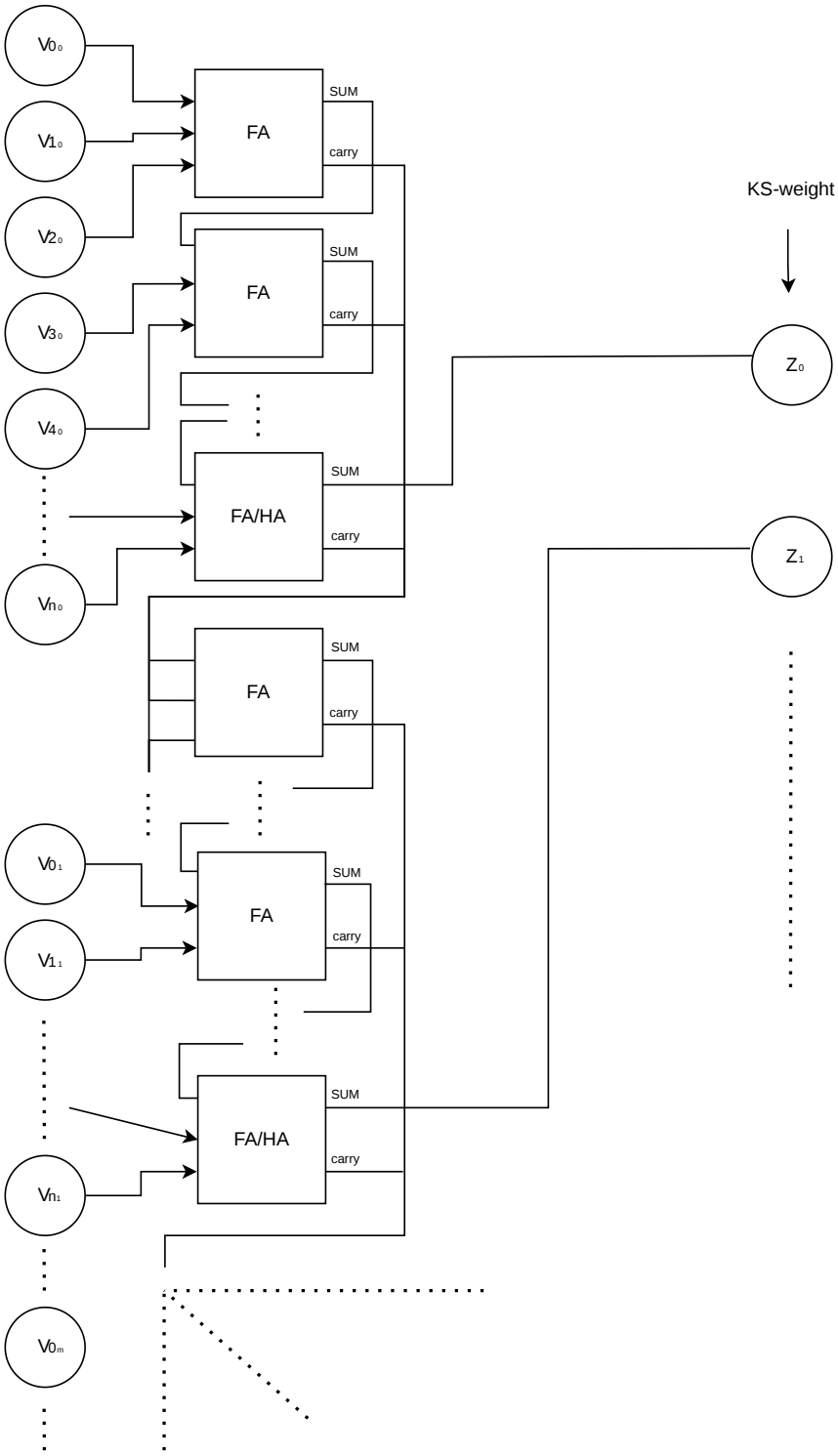

Figure A.1: CFA scheme for the KP, showing how the bitwise item weights V_{ij} are arranged by bit position and combined into the knapsack-weight bits Z_k by binary addition using HAs and FAs.

```
BQM ← dimod.BQM();
for i ← 0 to n − 1 do
    weight ← problem["weights"][i];
    for j ← 0 to b − 1 do
        if j-th bit of weight is 1 then
            create variable V_{i,j} and append it to columns[j];
        end
    end
end
counter ← 0;
for j ← 0 to max_columns − 1 do
    while |columns[j]| ≥ 3 do
        remove three variables from column j;
        create a Full Adder with removed variables;
        (λ, counter) ← lambdafunction(weights, j, counter);
        scale the Full Adder with λ and add it to BQM;
        append sum in columns[j] and carry in columns[j + 1];
    end
end
counter ← 0;
for j ← 0 to max_columns − 1 do
    collect the remaining variables in column j and an optional
     carry-in;
    counter ← counter · 2;
    if one variable remains then
        map the variable to Z_j;
    end
    else if two variables remain then
        create a Half Adder;
        (λ, counter) ← lambdafunction(weights, j, counter);
        scale the Half Adder with λ and add it to BQM;
        store the sum in Z_j and the carry to column j + 1;
    end
    else if three variables remain then
        create a Full Adder;
        (λ, counter) ← lambdafunction(weights, j, counter);
        scale the Full Adder with λ and add it to BQM;
        store the sum in Z_j and the carry to column j + 1;
    end
end
```

Algorithm A.1: Construction of the CFA QUBO, showing how item-weight bits are collected into columns, reduced with full adders, finalized with half or full adders, and combined with the capacity constraint.