npj | quantum information

Article

Published in partnership with The University of New South Wales

https://doi.org/10.1038/s41534-025-01097-8

A quantum algorithm for solving 0-1 Knapsack problems

Sören Wilkening[1,2] ✉, Andreea-Iulia Lefterovici[1], Lennart Binkowski[1], Michael Perk[3], Sándor P. Fekete[3] & Tobias J. Osborne[1]

We present two novel contributions for achieving and assessing quantum advantage in solving difficult optimisation problems, both in theory and foreseeable practice. (1) We introduce the "Quantum Tree Generator" to generate in superposition all feasible solutions of a given 0-1 knapsack instance; combined with amplitude amplification, this identifies optimal solutions. Assuming fully connected logical qubits and comparable quantum clock speed, QTG offers perspectives for runtimes competitive to classical state-of-the-art knapsack solvers for instances with only 100 variables. (2) By introducing a new technique that exploits logging data from a classical solver, we can predict the runtime of our method way beyond the range of existing quantum platforms and simulators, for benchmark instances with up to 600 variables. Under the given assumptions, we demonstrate the QTG's potential practical quantum advantage for such instances, indicating the promise of an effective approach for hard combinatorial optimisation problems.

In recent years, the prospect of real-world quantum computing has raised hopes for solving hard combinatorial optimisation problems, leading to tremendous theoretical work on developing and analysing suitable quantum methods. On the one hand, it is reasonable to expect in the next years that quantum information processing devices will comprise many thousands of physical qubits[1]. On the other hand, this progress is inhibited by two major challenges.

At the lower level, one challenge emerges from the effects of decoherence[2,3]. The current error rates experienced by all extant quantum devices place them well above the fault tolerance threshold[4] required for the effective deployment of quantum error correction[5]. Further critical roadblocks, depending on platform[6–14], also include the difficulties of implementing mid-circuit measurements[15] and entangling gates[16]. That quantum computers deliver useful improved and accelerated solutions is expected in the long term when logical qubits are plentiful and cheap. However, in the near- and mid-term, the situation is subtle and nuanced. Justifying the considerable effort for ensuing research and development hinges on the expected practical usefulness of quantum devices.

At a higher level, the challenge lies in demonstrating quantum advantage in practical applications. In recent times, there has indeed been considerable optimism that combinatorial optimisation problems provide a rich class of application areas whose solutions quantum computers will improve[17]. Although quantum devices have already demonstrated accelerated solutions for synthetic problems[18], there has been, to date, no conclusive demonstration of a practical quantum advantage for realistic instances in comparison with the *best classical approach*. Quantum heuristics[19] currently do not provide competitive results when compared with classical solvers, such as CPLEX[20], GUROBI[21] and CP-SAT[22] which routinely solve problems involving thousands of variables to *provable optimality*.

On the quantum side, the development of new algorithms has been almost exclusively driven by theoretical measures, in particular by worst-case, asymptotic runtime analysis. In particular, Grover's algorithm[23] and its provable quadratic speedup over unstructured linear search have spawned a rich manifold of quantum routines[24–27]. However, these theoretical insights have to be taken with a grain of salt when it comes to their practical applicability: A favourable asymptotic worst-case complexity does not necessarily imply a speedup for instance sizes that are relevant in practice. This already holds true for the comparison of two purely classical algorithms. The simplex method[28] for solving linear programmes has a worst-case runtime exponential in the input dimension, but remains one of the best-performing algorithm in practice. In contrast, the ellipsoid method[29] with worst-case runtime polynomial in the input dimension, performs poorly in practice in most cases, especially when compared to the simplex method[30]. This immanent gap between asymptotic worst-case behaviour and practical performance does not necessarily shrink when the compared algorithms run on completely different architectures.

Thus, we are faced with a fundamental dilemma when exploring the possible real-world impact of quantum methods for practical optimisation: On the one hand, the justification for building and tuning quantum devices hinges on their practical impact; on the other hand, gauging their usefulness

[1]Institut für Theoretische Physik, Leibniz Universität Hannover, Hannover, Germany. [2]Volkswagen AG, Wolfsburg, Germany. [3]Institut für Betriebssysteme und Rechnerverbund, Technische Universität Braunschweig, Braunschweig, Germany. ✉e-mail: soeren.wilkening@itp.uni-hannover.de

by running real-world benchmarks is impossible before these devices exist. A possible recourse for the latter is to perform classical simulations of quantum methods; however, possibilities for general gate-based simulations are saturated both in memory and runtime at around 50 qubits[31], which is far too few to evaluate the performance of a quantum algorithm against realistic benchmark instances (which typically would require between 100 and 10,000 qubits).

One recent approach[32] for obtaining realistic estimates for runtimes of quantum algorithms beyond asymptotic scaling is to derive quantitative complexity bounds for quantum routines, rather than asymptotic worst-case $\mathcal{O}$-expressions. These bounds are fed with data obtained by running classical versions of these routines. In this light, a recent study[33] was able to show that the quantum version of the simplex method[34], despite its promising asymptotic runtime, cannot outperform the standard simplex method on any reasonable benchmark instance. While this framework is powerful for benchmarking quantum routines when having formulas for the expected runtime, it does not provide enough tools for analysing quantum algorithms.

In this paper, we directly take on the twin challenges of (1) developing a quantum method to solve combinatorial optimisation problems and (2) evaluating the performance of this method against benchmark instances. We target the knapsack problem as a most basic optimisation problem whose difficulty is based on a single linear constraint, which frequently appears in real-world applications such as portfolio optimisation and securitisation[35], and often arises as a subproblem in more complex problems such as resource allocation and scheduling. Despite being NP-hard, the knapsack problem has allowed the practical computation of provably optimal solutions for instances of interesting size. This still does not make it an easy problem: A recently proposed set of benchmark instances[36] is challenging both exact solvers and heuristics.

For this fundamental problem, we develop a quantum method we name the Quantum Tree Generator (QTG), which combines classical structural insights with properties of quantum algorithms to first generate all feasible solutions in superposition, from which an optimal solution is extracted with quantum amplitude amplification[24]. This approach is based on a number of concepts from classical algorithmics: polyhedral combinatorics (which provide a characterisation of the set of feasible solutions), output-sensitive algorithms (as the set of feasible solutions may be significantly smaller then the set of all solutions) and randomised algorithms. This is combined with insights from quantum computing: the linear structure of feasible solutions (which allows it to compute a superposition in time that is linear in the number of variables) and amplitude amplification.

While the main objective of our work is to demonstrate the potential practical perspectives, it is worth noting that it is not a mere heuristic (such as a greedy approach or a local search), but comes with some theoretical performance guarantees at the expense of an exponential worst-case runtime: As highlighted by Dürr and Høyer, the quantum exponential searching algorithm (a generalisation of Grover, i.e., amplitude amplification algorithm) in combination with thresholding finds the index of the optimum value within $O(\sqrt{N})$ probes with probability at least $\frac{1}{2}$ (see Theorem 1 in[37]). Thus, running the quantum maximum finding routine (QMaxSearch) for $\leq \sqrt{2^n}$ iterations, makes QTG similar to an exact algorithm that finds an optimum with high probability, making a comparison with classical exact algorithms both fair and interesting. As with all exact algorithms for NP-complete problems, this performance guarantee comes at the expense of an exponential worst-case runtime on the theoretical side. In this way, it resembles any classical exact algorithm that may also require an exponential worst-case runtime.

However, just like for classical algorithms, this does not rule out *practically* useful runtimes: Can we achieve good (or even optimal) results in reasonable time that can (in principle, with the standard disclaimers to account for technical developments of future quantum devices) compete with classical algorithms on relevant benchmark instances, instead of purely theoretical asymptotic worst-case running times? We aim for this by limiting the maximum number of Grover iterations to a polynomial M (as discussed in Section "Methods"), which ensures reasonable runtimes at the expense of theoretical performance guarantees. Although we cannot *guarantee* finding OPT with high probability, we observe high probabilities on all tested instances. This makes QTG also useful as a heuristic for a range of relevant instances of interesting size, which is the main focus of our work. We explore these perspectives by a comparison with classical exact algorithms: These are run with a comparable time limit with the objective of finding good solutions, *without the need to establish optimality*.

In order to evaluate the performance of the QTG-based search on realistic benchmark instances, we also introduce a novel technique to compute the quantum algorithm's runtime: Extracting crucial logging data from the classical solution, obtained via the current champion COMBO[38] method, we can infer the expected number of cycles required by the QTG-based method to solve the same instance. Our benchmarking leads us to predict that the QTG-based method requires fewer cycles to solve realistic instances already at 100 variables. It does so without huge demands on memory, as it operates entirely in the circuit model, only requiring the working qubits as space requirements. In contrast, the COMBO method – based in part on dynamic programming – can make huge demands on memory, with 10^{10} bits being routinely requested. The QTG-based search, however, requires only a constant multiple of the variable number (i.e., the number n of items) in logical qubits. This indicates a *possible* quantum advantage in both time and space starting at as few as 100 variables, offering a clear perspective of a *practical* quantum advantage for instance sizes of real-world relevance, once fully connected logical qubits and improved quantum clock speeds are available.

While converting this potential on actual quantum hardware hinges on dealing with a spectrum of hardware issues, it is also fair to point out that the speed of current classical solvers rests on decades of development, both in hardware and algorithms: As highlighted by Bixby[39] in the context of Linear Programming, 15 years of progress were sufficient to achieve a speedup of six orders of magnitude for classical solvers, with innovations in hardware and algorithms each contributing with three orders of magnitude. Thus, our main contribution is not an immediate practical quantum advantage, but a path that seems promising enough to justify further developments in real-world quantum hardware in pursuit of such an advantage.

In 1995, Pisinger developed the branch-and-bound approach called EXPKNAP[40]. At the time, the algorithm stood out as one of the most efficient algorithms documented in literature for the 0-1 knapsack problem (0-1-*KP*). In recent years, integer programming (IP) and constraint programming (CP) solvers emerged as the industry standard for various optimisation problems. The integer programming solver GUROBI (based on branch and cut methods) and CP-SAT[22] (portfolio solver based on constraint programming) are popular solvers in their respective field. In 1999, the COMBO algorithm emerged as a dynamic programming solver specifically designed for 0-1-*KP*. Known for its stable behaviour due to its pseudo-polynomial time complexity bound[38], it stands as the current leading solver for this problem.

Quantum methods to tackle constrained optimisation (and closely related satisfaction) problems have been previously studied in works such as nested quantum search[25] and quantum branch-and-bound[27,41]. Nested quantum search is an enhancement over Grover's[23] unstructured search algorithm for solving constrained satisfaction problems. It consists of applying Grover's search on a sub-problem to identify potential solutions, followed by another Grover's search application on the remaining variables to extract the actual solutions from the set of potential solutions. This prepares a uniform superposition over the set of feasible assignments. The number of iterations for the nested quantum search to find an optimal solution is $\mathcal{O}(\sqrt{d^{\alpha}})$, where d is the dimension of the search space, and $\alpha < 1$ is a constant depending on the nesting depth. Quantum branch-and-bound (QBnB) uses quantum backtracking[26,42] as a subroutine. The algorithm improves upon nested quantum search by searching at each level of the tree only for partial assignments that satisfy the constraints and show promise to deliver optimal solutions based on unassigned variables. The number of

iterations for the quantum branch-and-bound methods to find an optimal solution is $\mathcal{O}(\sqrt{Tp})$, where T is the number of nodes and p is the depth of the tree. Both algorithms achieve a quadratic speedup over their respective classical counter parts. We benchmark these methods in Supplementary Information F.

In contrast, our method prepares a superposition of all feasible states by sequentially imposing the constraints of an instance (as shown in Fig. 1), followed by an augmented version of amplitude amplification to get an optimal solution.

Further quantum algorithmic approaches to solve combinatorial optimisation problems are quantum annealing[43] and the closely related quantum adiabatic algorithm[44]. In both approaches, the initial uniform superposition of all (not necessarily feasible) assignments evolves w.r.t. a time-dependent Hamiltonian, encoding the classical objective function and constraints, whose strength is varied over time. If the rate of change is sufficiently slow, the system is guaranteed to remain close to a ground state of the instantaneous Hamiltonian. The ground states of the final Hamiltonian then correspond to optimal solutions to the encoded combinatorial optimisation problem. A discretized version of the quantum adiabatic algorithm is given by the quantum approximate optimisation algorithm[45] and its generalisation to the quantum alternating operator ansatz (QAOA)[46]. Here, the continuous time evolution is replaced with the p-fold alternating time evolution with the initial Hamiltonian (usually called the *mixer*) and the target Hamiltonian (the *phase separator*) where the respective evolution times enter as parameters to be optimised. In the limit $p \to \infty$, if the angles are chosen appropriately (e.g., to mimick the quantum adiabatic algorithm), QAOA is guaranteed to find an optimal solution[47]. In their original proposal, Farhi et al.[45] established a guaranteed approximation ratio of 0.6924 for depth-1-QAOA applied to the MAX-CUT problem. To our knowledge, there exist no similar guarantees for QAOA (and variations such as in[19]) when applied to the knapsack problem.

Results

Setting

We demonstrate the capabilities of our QTG-based search on a diverse set of benchmark instances. All experiments were carried out on a regular desktop workstation with an AMD Ryzen 7 5800X (8 × 3.8 GHz) CPU and 128 GB of RAM. Instances were produced using a state-of-the-art instance generator from Jooken et al.[36] that was made available in[48]. The problem instances involve items partitioned into groups with exponentially decreasing profits and weights. Items within the same group have similar, slightly perturbed profits and weights. The last group differs by having uncorrelated small profits and weights, introducing a variety of profit-weight ratios. The generator created six instances for every value $2 \le g \le 10$ (number of groups) and size $n = 50, 100, \ldots, 600$. The parameter g is an input for the instance generator, which defines how many groups of similar profits/weights are desired, where a larger g values typically correspond to more difficult instances Jooken et al.[36]. All instances were generated with capacity $c = 10\,000\,000\,000$. This yielded a total of 648 benchmark instances with various difficulties. In Supplementary Information A, we compare the performance of a wide range of classical solvers on a separate benchmark set. When it comes to exact solvers that are able to find an optimal solution and simultanously prove its optimality, COMBO is the best-performing classical solver for our benchmark set. CP-SAT and GUROBI are also able to find an optimal solution for all instances, but they were often not able to prove optimality within the given time limit of 300s. Thus, for the remainder of this section, we use the runtime of COMBO as a reference and set it as an upper time limit for the classical solvers GUROBI and CP-SAT.

Space/memory requirement

First, we compare the qubit counts for QTG-based search and for the quantum branch-and-bound (QBnB) method proposed by Chakrabarti et al.[41] with the memory requirements of COMBO, GUROBI, and CP-SAT (see Fig. 2). The required memory (resident set size) was measured with GNU time utility. In contrast to quantum systems, classical computers use a certain amount memory for storing the code of the programme. To calculate the data memory usage, all solvers were executed on tiny instances (10 items). This makes sure that all relevant code memory pages were loaded by the solvers at some point and, therefore, detected by the GNU time utility. The data memory usage is now calculated by subtracting the minimum memory usage of that solver in the preliminary experiment. We observe that, assuming the comparability of bits and qubits, both the QTG-based method and QBnB have drastically lower space requirements than the RAM-based classical solvers. While the number of QTG's working qubits stays constant for instances of the same size, COMBO's memory requirements increase exponentially for harder instances. CP-SAT and especially GUROBI demand a similarly high amount of memory, even though both algorithms were not always able to prove optimality within the given time limit which potentially avoids a portion of their memory requirements. The huge gap between space requirements of the circuit-based quantum methods and the classical RAM-based algorithms is ultimately no surprise. Other quantum algorithms such as Grover search or QBnB also have a similar advantage over classical RAM-based algorithms. However, we can confidently rule out hidden space-time trade-offs of QTG-based search that would diminish the next observations.

Cycle comparison

We now compare the classical cycle counts of COMBO and CTG-based search (measured via the time stamp counter (TSC) values, that are present on x86 processors) with the quantum cycle counts of QTG-based search and

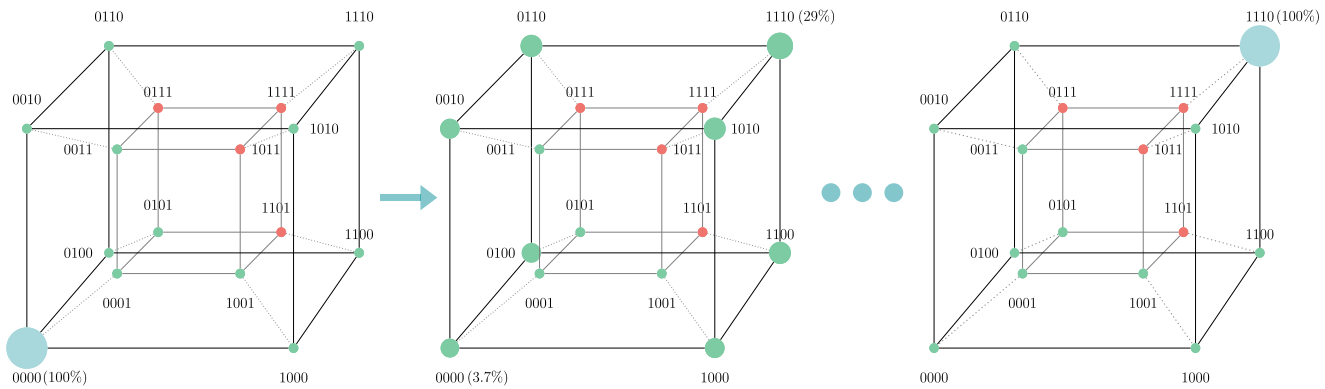


Fig. 1 | The action of the QTG on the $KP_4(p, w; 7)$ example. The path register holds 16 computational basis states—here depicted as the corners of a hypercube—representing all feasible (green dots) and infeasible (orange dots) four-item paths. The state $|1110\rangle$ represents an optimal solution. The initial state of the system is $|0000\rangle$, corresponding to an entirely empty knapsack. After one application of the QTG, an optimal state's sampling probability is increased to 29%. After 12 applications of the QTG, an optimal state is reached.

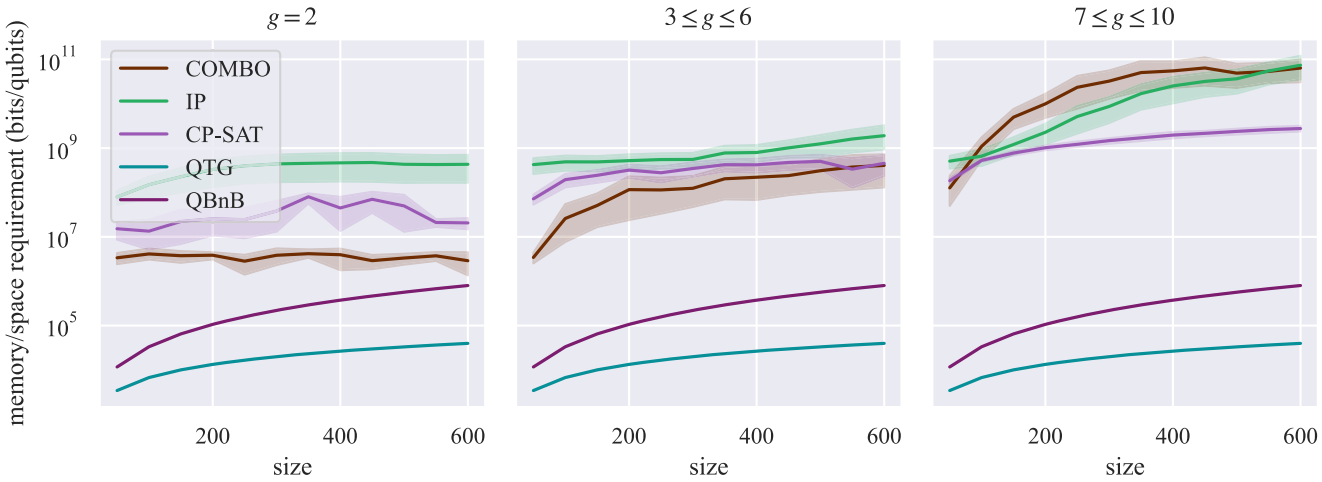


Fig. 2 | Memory/space requirements for quantum methods quantum branch-and-bound (QBnB) and QTG-based search (QTG) in comparison with standard classical solvers (COMBO, GUROBI, and CP-SAT). The subfigures (from left to right) show the measured average requirements for different groups $g = 2$, $3 \le g \le 6$ and $7 \le g \le 10$, respectively. See Section "Results" for a description of the instance set. The time limit for GUROBI and CP-SAT was set to COMBO's execution time. As a result, these solvers were unable to find an optimal solution for all benchmark instances. The blue line shows the number of qubits required for executing QTG-based search for the given instances. Its main qubit contribution is due to the augmented quantum amplitude amplification; all the ancilla qubits for gate decompositions etc. are accounted for. Assuming for the moment comparability of bits and qubits, the initial requirement for COMBO is four orders of magnitude higher than for QTG-based search. In the regime of hard instances, i.e., $7 \le g \le 10$, the memory demands of COMBO quickly rise to 10 GB, while QTG-based search as well as QBnB retain their moderate qubit requirements. Note that this is a feature of quantum algorithms, so methods such as Grover search have a similar advantage over classical RAM-based algorithms.

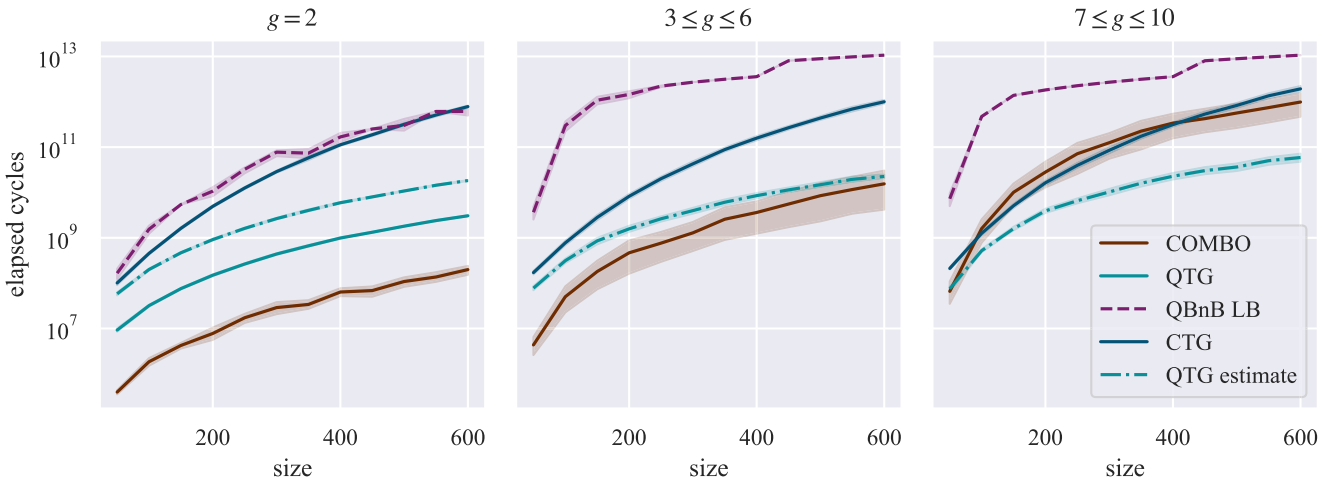


Fig. 3 | Comparison of quantum methods (QBnB and QTG-based search) with the classical methods COMBO and CTG. The subfigures (from left to right) show the measured average performance for different groups $g = 2$, $3 \le g \le 6$ and $7 \le g \le 10$, respectively. See Section "Results" for a description of the instance set. Note that the values for QTG are realistic estimates, based on simulation (for $g = 2$) and calibrated estimates (for $g \ge 3$); on the other hand, values for QBnB are lower bounds, based on several benevolent assumptions, so a realistic runtime can be expected to be higher; see Supplementary Section F 2. For larger values of g, computational effort for full QTG simulation is excessive, so we provide estimates for QTG derived from CTG and justified in Supplementary Section B1. Assuming comparability of classical and quantum cycles, we observe within the studied regimes that QTG-based search starts outperforming COMBO on instances with $7 \le g \le 10$. The trend continues towards larger instances and indicates a generally more favourable slope for the quantum method than for COMBO.

QBnB, see Fig. 3. Note that GUROBI and CP-SAT heavily rely on parellelisation, which makes it difficult to count the exact number of cycles in a meaningful way. When it comes to CP-SAT (a portfolio solver), it would destroy the purpose of the portfolio approach when we would limit the number of threads to one. We instead set the number of available cores for both GUROBI and CP-SAT to 8 (i.e., the number of cores of the workstation). Despite both solvers utilising multiple threads, they were still outperformed by COMBO in terms of runtime. Thus, we decided to use COMBO as a reference for the classical solvers in this comparison.

Because QTG-based search is a non-deterministic algorithm, an optimal solution is only sampled with a certain success probability recorded for every instance. For the easier instances with $g = 2$, we employ both the exact simulation for QTG, as well as a hybrid version (QTG estimate), for which we execute the CTG with the square of the prescribed number of Grover iterations to determine the success probability. For harder instances, we omit the exact simulation due to long processing times, but keep the QTG estimate (see Supplementary Section B 1), which shows the same scaling behaviour as the exact calculator but with a positive offset, making it a good estimate. We carry out 100 applications of the QTG-based QMax-Search with a bias $b = n/4$ and $M = 700 + n^2/16$ Grover iterations. As shown in Fig. 4, the success probabilities of finding an optimal solution for instances with difficulty $2 \le g \le 6$ are above 80%, while for hard instances with

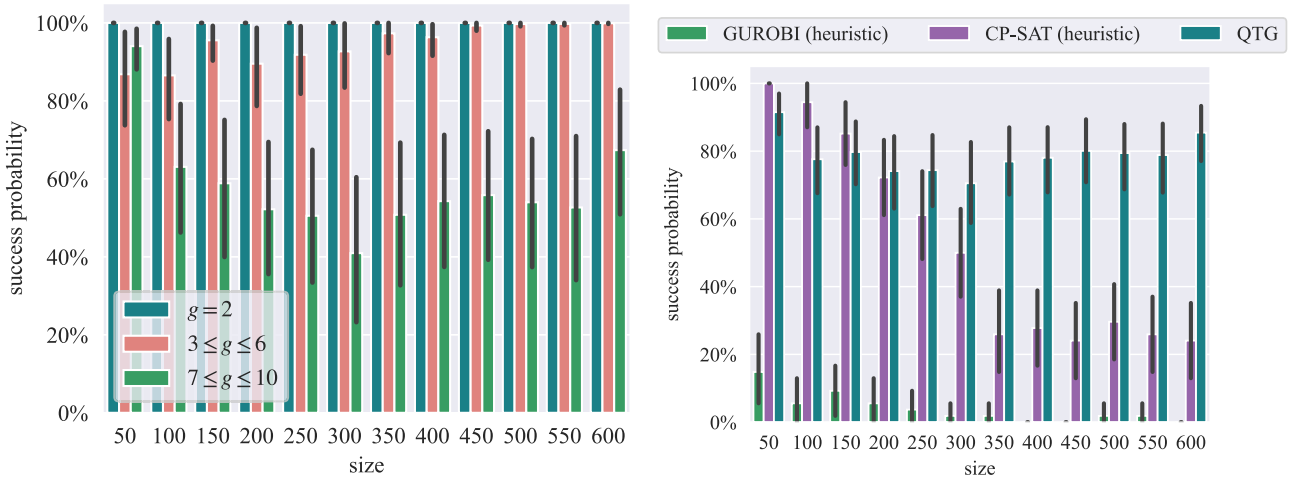


Fig. 4 | Success probabilities of finding optimal solutions for the QTG-based search and classical heuristics. (Left) For instances of difficulties ranging from $g = 2$ to $g = 10$. On average, for instances with difficulties $2 \leq g \leq 6$, the success probabilities are above 80%, while for hard instances they are above 40%. (Right) For the classical solvers GUROBI and CP-SAT which were given an optimal solution as an initial bound and fine-tuned to focus on finding good solutions. COMBOs runtime on the specific instance was set as a time limit leaving finding an optimal solution as the only task (thus labelling both solvers as heuristics).

$7 \leq g \leq 10$ are above 40%. The quantum cycles and number of qubits required to run QTG were calculated as described in Section "Methods" and averaged over all runs. All this provides realistic evaluations for QTG, based on exact simulations (for easier instances up to 600 items) and calibrated pessimistic estimates (for harder instances up to 600 items). This differs from the runtime estimates for QBnB, which are generous lower bounds for any practical runtime, based on several benevolent assumptions; a realistic runtime of QBnB can be expected to be considerably higher. These assumptions for QBnB (which are *not* granted to QTG) include a success probability of 1 for all quantum subroutines; see Supplementary Section F 2 for details.

As the size of the tree grows exponentially, we limit the maximum number of tree nodes to 10^{10} giving us an even more optimistic lower bound on the actual number of cycles. We numerically observe that, for hard instances ($g \geq 7$), COMBO's cycle demand overtakes the quantum method's elapsed cycles already at instances with 100 variables and indicates a steeper slope overall. Meanwhile, the optimistic lower bound for QBnB's cycle count remains at least one order of magnitude above COMBO's cycle count throughout all instances. Although the improved performance of QTG-based search for harder instances in comparison to COMBO certainly is an impressive result, it has to be taken with a grain of salt; while COMBO, by design, returns an optimal solution with certainty, QTG-based search cannot guarantee optimality and occasionally fails to deliver an optimal solution. Nonetheless, QTG was able to yield an optimal solution with reasonably high probability (≥40%). This indicates a high probability that an optimal solution will be found within only a few applications of QTG, i.e., > 90% after five and > 99% after ten QTG runs.

Heuristic comparison

As QTG does is not guaranteed to find an optimal solution, we conducted a second experiment comparing its performance to GUROBI, and CP-SAT where both solvers were given the objective value of an optimal solution as an initial bound. This way, both solvers only had to find an optimal solution without proving its optimality. The cycle counts of GUROBI and CP-SAT were measured using the TSC values, that are present on x86 processors. As both algorithms make heavy use of parallelisation the actual cycle counts can be larger by some constant factor. We set the time limit for GUROBI and CP-SAT to COMBO's execution time. Additionally, GUROBI was instructed to speed up the search for solutions by setting the `MIPFocus` parameter to 1. We also tested the integer greedy algorithm as well as the Monte Carlo tree search[49]. As these heuristics failed to deliver any optimal solution for the considered instances, we do not include them in the comparison.

Figure 4 shows how often both solvers were able to find an optimal solution without proving their optimality within the runtime of COMBO on the given instance; see Fig. 5 for the corresponding cycle counts. For all instance sizes CP-SAT was able to find an optimal solution more reliably than GUROBI with success probabilities of < 50% for instances of size ≥350. This result has stronger practical implications than the previous one: Introducing QTG-based search as a heuristic within existing branch-and-bound frameworks could markedly speed up the generation of an optimal solution, leaving the proof of optimality to the classical components.

On the other hand, Fig. 6 provides the outcomes of a comparison of QTG and COMBO, i.e., the respective success probabilities for finding optimal solutions within comparable time. (Due to the nature of COMBO, which is based on enhanced Dynamic Programming, this amounts to finishing a run within the alotted time.) Assuming comparability of classical and quantum cycles, we observe within the studied regimes that QTG-based search starts outperforming even the heuristic versions of classical solvers on instances with $7 \leq g \leq 10$.

Discussion

In this paper, we have proposed and investigated the Quantum Tree Generator (QTG), a quantum method for solving instances of the 0-1 knapsack problem. The QTG-based search emerges as a highly problem-specific version of quantum amplitude amplification with improved incorporation of the underlying classical structure at hand. Due to our novel runtime calculator, we were able to collect numerical data, rather than mere extrapolation, for benchmark instances of various difficulty with up to 600 variables.

On the evaluated benchmarks, the quantum method admitted favourable runtimes for hard instances with at least 100 variables. In addition to these runtime perspectives, it is worth mentioning another factor that often limits the performance of classical algorithms, even before a timeout: This is the potentially enormous memory consumption, in particular for methods based on Dynamic Programming, which is the basis of special-purpose methods such as COMBO. Such limitations do not play a role for quantum algorithms such as QTG or Grover's algorithm. In summary the presented results hint at a possible *practical* quantum advantage for solving 0-1-*KP* instances.

It is essential to acknowledge that there exist several other potential ways for improvement. These areas of potential enhancement encompass

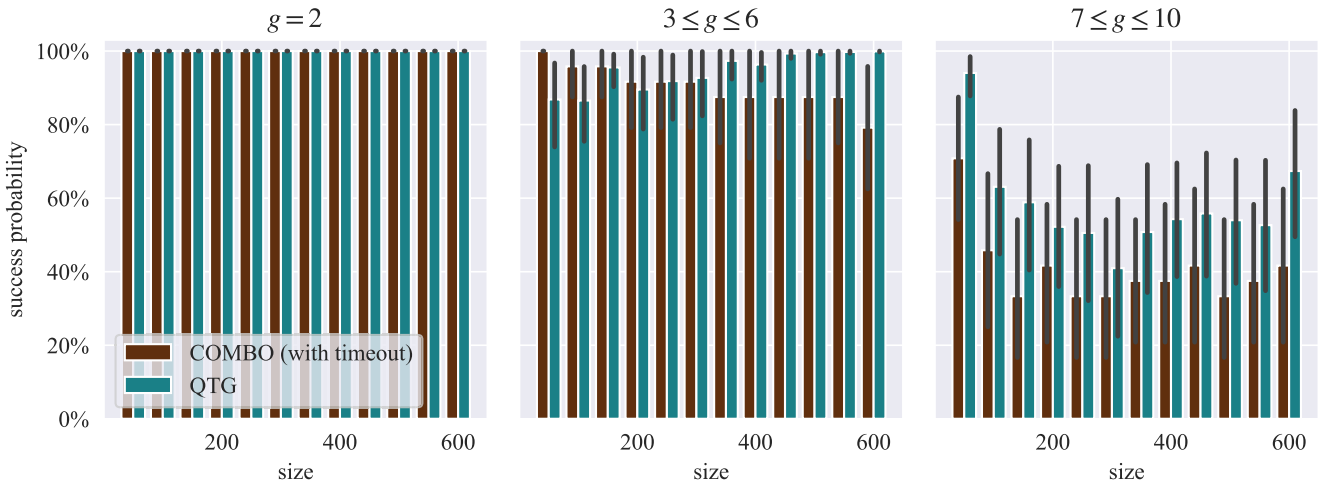


Fig. 5 | Comparison of QTG-based search with the classical solvers GUROBI and CP-SAT for different numbers of groups g where the classical solvers are given an optimal solution as an initial bound and COMBOs runtime as a time limit. See Section "Results" for a description of the instance set. Note that the values for QTG are realistic estimates, based on simulation (for $g = 2$) and calibrated estimates (for $g \geq 3$). For larger values of g, computational effort for full QTG simulation is excessive, so we provide estimates for QTG derived from CTG and justified in Supplementary Section B 1. Assuming comparability of classical and quantum cycles, we observe within the studied regimes that QTG-based search starts outperforming even the heuristic versions of classical solvers on instances with $7 \leq g \leq 10$. Note that the solvers have different success probabilities wihtin the given runtime, see Fig. 4.

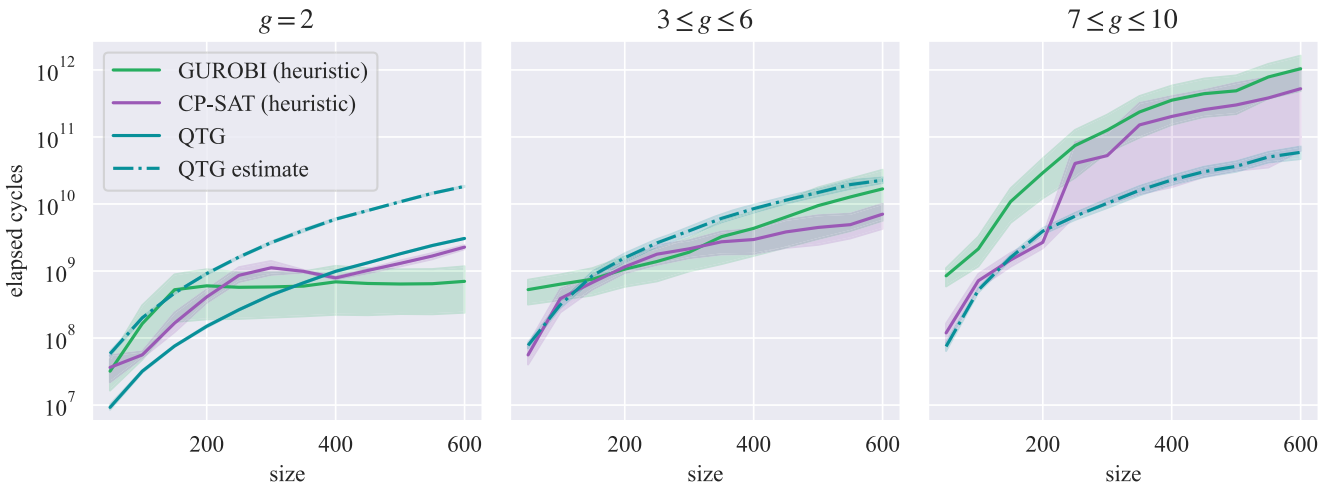


Fig. 6 | Success probabilities of COMBO when executed for the same number of cycles as QTG in comparison with those obtained by QTG. The subfigures (from left to right) show the measured average success probability for different groups $g = 2$, $3 \leq g \leq 6$ and $7 \leq g \leq 10$, respectively. Due to the nature of COMBO, it either produces an optimal solution or fails to find any solution. The results show that when both methods are executed with a timeout, QTG-based search is able to compete with COMBO when comes to finding optimal solutions for instances with $7 \leq g \leq 10$ with a high success probability.

the utilisation of upper-bound heuristics within the QTG to reduce branching, or the utilisation of a preprocessing branch-and-bound procedure, such that the QTG-based search is executed only on the remaining subsequent problem, or to use a different bias function based on the weight of the items.

Methods

0-1 Knapsack problem

Consider a set of n items, each of which has some integer profit $p_m > 0$ and weight $w_m > 0$. The objective is to select a subset of items of maximum cumulative profit, so that their cumulative weight do not exceed some threshold $c > 0$). An instance of the 0-1-KP is completely defined by the number n of items, a list $\boldsymbol{p} := (p_1, \ldots, p_n)$ of profit values, a list $\boldsymbol{w} := (w_1, \ldots, w_n)$ of weights, and the capacity c; we address a given instance in the following by $KP_n(\boldsymbol{p}, \boldsymbol{w}; c)$. It has the following formulation as an Integer Linear Programme (ILP):

$$\begin{aligned} &\text{maximise} \sum_{m=1}^{n} p_m x_m \\ &\text{subject to} \sum_{m=1}^{n} w_m x_m \leq c \\ &x_m \in \{0,1\}, \quad m = 1, \ldots, n. \end{aligned} \tag{1}$$

For the 0-1-KP, the binary variable x_m encodes the choice of either including item m into the knapsack ($x_m = 1$) or omitting it ($x_m = 0$). Therefore, a complete assignment of all items (also called *path*) constitutes a bit string of length n.

The standard procedure to formulate the 0-1-KP for study via a quantum computer is to assign one qubit to each decision variable x_m, thus representing the paths $\boldsymbol{x} = x_1 \ldots x_n$ as computational basis states $|\boldsymbol{x}\rangle^1$ in an

n-qubit *path register* $\mathcal{H}_1 = \mathbb{C}^{2^n}$. Additionally, for any feasible assignment $\boldsymbol{x}$, we store the binary representation of the remaining capacity $c - \sum_{m=1}^{n} w_m x_m$ as a quantum state $|c_{\boldsymbol{x}}\rangle^2$ in a $|c_{\text{bin}}|$-qubit *capacity register* $\mathcal{H}_2$, with $|c_{\text{bin}}| = \lfloor \log_2 c \rfloor + 1$. The total profit is stored as a state $|P_{\boldsymbol{x}}\rangle^3$ in a $|P_{\text{bin}}|$-qubit *profit register* $\mathcal{H}_3$, where P is any upper bound on the optimal profit. Furthermore, we optimise the depth of the circuit by using a $\max(n, |c_{\text{bin}}|, |P_{\text{bin}}|)$-qubit ancilla register $\mathcal{H}_a$. The total number of used qubits in the composite register $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \mathcal{H}_3 \otimes \mathcal{H}_a$ is therefore given by

$$n + |c_{\text{bin}}| + |P_{\text{bin}}| + \max(n, |c_{\text{bin}}|, |P_{\text{bin}}|). \tag{2}$$

Quantum tree generator

Our quantum algorithm for solving 0-1-KP operates entirely on the aforementioned composite register. The quantum routine is fundamentally based on two main ingredients:

1. a state preparation circuit that creates a superposition of all valid paths along with their total profit and remaining capacity as composite quantum states.
2. an augmented quantum amplitude amplification[24] which increases the success probability of measuring an optimal solution while avoiding endless loops.

The entire state preparation circuit, the Quantum Tree Generator (QTG) $\mathcal{G}$, arises as a cascade of *nlayer unitaries* U_m, i.e., $\mathcal{G} = \prod_{m=1}^{n} U_m$. Applying the QTG to the initial state $|\mathbf{0}\rangle^1 |c\rangle^2 |0\rangle^3$ generates a (generally non-uniform) superposition of all valid paths. The QTG iteratively includes/excludes all items in a weighted superposition while checking for feasibility and updating the capacity and profit registers. First, for every path, it is checked whether the remaining capacity covers the weight of the current item m. Next, every path that can include the item will be branched with a certain bias b using an Ry gate which we denote as $\mathrm{H}_b^{x_m}$, defined as

$$\begin{aligned} \mathrm{H}_b^0 &= R_y\left(2\,\mathrm{acos}\sqrt{\frac{1+b}{2+b}}\right) = \frac{1}{\sqrt{2+b}}\begin{pmatrix} \sqrt{1+b} & -1 \\ 1 & \sqrt{1+b} \end{pmatrix}, \\ \mathrm{H}_b^1 &= R_y\left(2\,\mathrm{acos}\sqrt{\frac{1}{2+b}}\right) = \frac{1}{\sqrt{2+b}}\begin{pmatrix} 1 & -\sqrt{1+b} \\ \sqrt{1+b} & 1 \end{pmatrix}, \end{aligned} \tag{3}$$

where x_m is the m-th bit of an intermediate solution $\boldsymbol{x}$. Upon successful item inclusion, the item's weight is subtracted from the capacity register's state and its profit is added to the profit register's state. Both, subtraction and addition, are conducted using Quantum Fourier Transform (QFT) adders[50]. In total, the unitaries U_m are comprised of three components $U_m = U_m^3 U_m^2 U_m^1$:

$$\begin{aligned} U_m^1 &= C^2_{\geq w_m}\left(\mathrm{H}_b^{x_m}\right) \\ U_m^2 &= C_m^1\left(\mathrm{SUB}_{w_m}\right) \\ U_m^3 &= C_m^1\left(\mathrm{ADD}_{p_m}\right). \end{aligned} \tag{4}$$

We denote the control of the biasing Ry gates with $C^2_{\geq w_m}$ because it asks whether the second register's state is greater or equal to the integer w_m. Subtraction and addition are controlled on the m-th qubit in the first register: C_m^1. The bias b in order to maximise the sampling probability of a path

Box 1 | Quantum Tree Generator

Input:

Knapsack instance $KP_n(\boldsymbol{p}, \boldsymbol{w}; c)$, feasible assignment $\boldsymbol{x}$

1. Initialize:

Initialize the three registers in the state $|0\rangle^1 |\mathbf{c}\rangle^2 |0\rangle^3$.

2. Tree traversing:

For each item $m = 1, \ldots, n$, perform the following steps:

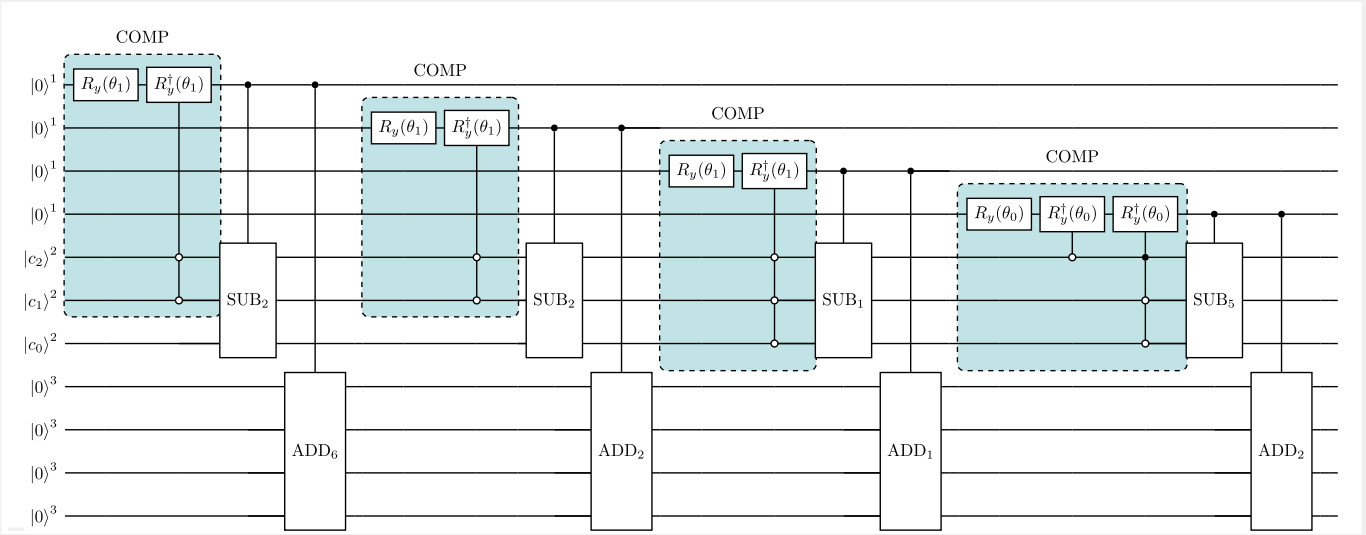
(a) Create a superposition. Apply $R_y(\theta_{0/1})$ on the m-th qubit in register 1, controlled on whether w_m does not exceed the remaining capacity stored in the register 2.

(b) Update the remaining capacities. Subtract w_m from the state of register 2, controlled on the m-th qubit in register 1.

(c) Update the total profit. Add p_m to the state of register 3, controlled on the m-th qubits in register 1.

Example:

Knapsack instance $KP_4(\boldsymbol{p}, \boldsymbol{w}; 7)$ with weights $\boldsymbol{w} = (2, 2, 1, 5)$ and profits $\boldsymbol{p} = (6, 2, 1, 2)$. Here $\theta_1 = 2\mathrm{acos}\sqrt{\frac{2}{3}}$ and $\theta_0 = 2\mathrm{acos}\sqrt{\frac{1}{3}}$.



$\boldsymbol{x}$ within the superposition created by the QTG is given by

$$b_{opt} = \frac{n}{\Delta} - 2 \approx \frac{n}{\Delta}, \tag{5}$$

where $\Delta := \Delta(\boldsymbol{x}', \boldsymbol{x})$ is the Hamming distance of $\boldsymbol{x}$ to a given intermediate solution $\boldsymbol{x}'$. More details on the construction of the biasing gates $\mathrm{H}_b^{x_m}$ and unitaries U_m can be found in Supplementary Section B. The concept of the QTG is summarised in Box 1, along with the $KP_4(\boldsymbol{p}, \boldsymbol{w}; 7)$ circuit example.

The superposition created by the QTG is further manipulated in order to increase the probability of measuring an optimal solution state. For this, we employ quantum maximum finding (QMaxSearch)[37], which iteratively calls an amplitude amplification (QSearch)[24] routine. The intermediate best solution's total profit provides the current threshold and is therefore entirely evaluated on the third register. More details can be found in Supplementary Section C.

We further choose $\Delta = 4$, hence a bias of $b = n/4$, and a maximum number of $M = 700 + n^2/16$ Grover iterations within QSearch. These values for the hyperparameters are chosen to perform well in practice. In a classical preprocessing step, we sort the list of items by decreasing density (profit versus cost), and obtain an initial threshold for QMaxSearch by greedily packing as many items into the knapsack as possible.

By combining the application of the QTG (see Fig. 1 for the $KP_4(\boldsymbol{p}, \boldsymbol{w}; 7)$ example), the phase flip operator within QSearch, and quantum measurements with a detailed analysis of the resource requirements of every subcircuit, we calculate the number of necessary quantum cycles for obtaining an optimal solution with a certain success probability. More details can be found in Supplementary Section D and E. Our analysis makes the following crucial assumptions:

1. all qubits and gates are noiseless, logical components.
2. all single-qubit gates, singly-controlled single-qubit rotations, and Toffoli gates together constitute a universal set of elementary gates of equal implementation cost.
3. disjoint gates can be executed in parallel, constituting a single cycle on the QPU.
4. singly-controlled single-qubit rotations and Toffoli gates can be executed on arbitrary tuples of qubits.

Lastly, the QTG can be dequantised in order to yield a probabilistic classical algorithm: Starting from the all-zero string $\boldsymbol{0}$, iterate through all items/bits x_m and apply conditioned bit flips with probability $|(\mathrm{H}_b^{x_m})_{12}|^2$. The condition $C^2_{\geq w_m}$ can be realised classically. The entire algorithm then consists of sampling multiple times from the generated distribution of bit strings and updating the current solution whenever a better candidate was sampled. More details can be found in Supplementary Section B 1. We name this quantum-inspired method the *Classical Tree Generator* (CTG). Similar (although not quantum-inspired) constructions already exist in the literature[49].

Data availability

Data is available at https://doi.org/10.5281/zenodo.16895828.

Code availability

Code is available at https://doi.org/10.5281/zenodo.16895828.

Received: 19 November 2024; Accepted: 5 August 2025;
Published online: 26 August 2025

References

1. Gambetta, J. & Steffen, M. Charting the course to 100,000 qubits. https://research.ibm.com/blog/100k-qubit-supercomputer (2023).
2. Brandt, H. E. Qubit devices and the issue of quantum decoherence. *Prog. Quant. Electron* **22**, 257–370 (1999).
3. Schlosshauer, M. Quantum decoherence. *Phys. Rep.* **831**, 1–57 (2019).
4. Preskill, J. Quantum computing in the NISQ era and beyond. *Quantum* **2**, 79 (2018).
5. Lidar, D. A. & Brun, T. A. *Quantum Error Correction* (Cambridge University Press, 2013).
6. Kjaergaard, M. et al. Superconducting qubits: current state of play. *Annu. Rev. Condem. Ma. P.* **11**, 369–395 (2020).
7. Siddiqi, I. Engineering high-coherence superconducting qubits. *Nat. Rev. Mater.* **6**, 875–891 (2021).
8. Blatt, R. & Roos, C. F. Quantum simulations with trapped ions. *Nat. Phys.* **8**, 277–284 (2012).
9. Cirac, J. I. & Zoller, P. Quantum computations with cold trapped ions. *Phys. Rev. Lett.* **74**, 4091 (1995).
10. Häffner, H., Roos, C. F. & Blatt, R. Quantum computing with trapped ions. *Phys. Rep.* **469**, 155–203 (2008).
11. Leibfried, D., Blatt, R., Monroe, C. & Wineland, D. Quantum dynamics of single trapped ions. *Rev. Mod. Phys.* **75**, 281 (2003).
12. Madsen, L. S. et al. Quantum computational advantage with a programmable photonic processor. *Nature* **606**, 75–81 (2022).
13. Slussarenko, S. & Pryde, G. J. Photonic quantum information processing: a concise review. *Appl. Phys. Rev.* **6**, 041303 (2019).
14. Wu, B.-H., Alexander, R. N., Liu, S. & Zhang, Z. Quantum computing with multidimensional continuous-variable cluster states in a scalable photonic platform. *Phys. Rev. Res.* **2**, 023138 (2020).
15. Rudinger, K. et al. Characterizing midcircuit measurements on a superconducting qubit using gate set tomography. *Phys. Rev. Appl.* **17**, 014014 (2022).
16. Reagor, M. et al. Demonstration of universal parametric entangling gates on a multi-qubit lattice. *Sci. Adv.* **4**, eaao3603 (2018).
17. Ajagekar, A., Humble, T. & You, F. Quantum computing based hybrid solution strategies for large-scale discrete-continuous optimization problems. *Comput. Chem. Eng.* **132**, 106630 (2020).
18. Arute, F. et al. Quantum supremacy using a programmable superconducting processor. *Nature* **574**, 505–510 (2019).
19. van Dam, W., Eldefrawy, K., Genise, N. & Parham, N. Quantum optimization heuristics with an application to knapsack problems. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, 160–170 (IEEE, 2021).
20. IBM. IBM ILOG CPLEX Optimization Studio (2022).
21. Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual (2023).
22. Google OR-Tools. Google OR-Tools Documentation (2024).
23. Grover, L. K. A fast quantum mechanical algorithm for database search. Preprint at: https://doi.org/10.48550/arXiv.quant-ph/9605043 (1996).
24. Brassard, G., Høyer, P., Mosca, M. & Tapp, A. Quantum amplitude amplification and estimation arXiv: quant-ph/0005055 (2000).
25. Cerf, N. J., Grover, L. K. & Williams, C. P. Nested quantum search and structured problems. *Phys. Rev. A* **61**, 032303 (2000).
26. Montanaro, A. Quantum-walk speedup of backtracking algorithms. *Theory Comput.* **14**, 1–24 (2018).
27. Montanaro, A. Quantum speedup of branch-and-bound algorithms. *Phys. Rev. Res.* **2**, 013056 (2020).
28. Dantzig, G. B. Linear programming in problems for the numerical analysis of the future. In *Proc. Symposium on Modern Calculating Machinery and Numerical Methods, UCLA,* 29–31 (1948).
29. Khachiyan, L. G. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk* Vol. 244, 1093–1096 (Russian Academy of Sciences, 1979).
30. Ye, Y. *Wiley-Interscience Series in Discrete Mathematics and Optimization* 419–419 (John Wiley and Sons, Ltd, 1997).
31. Lykov, D., Shaydulin, R., Sun, Y., Alexeev, Y. & Pistoia, M. Fast simulation of high-depth QAOA circuits. In *Proc. SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, 1443–1451 (Association for Computing Machinery, 2023).
32. Cade, C., Folkertsma, M., Niesen, I. & Weggemans, J. Quantifying Grover speed-ups beyond asymptotic analysis. *Quantum* **7**, 1133 (2023).

33. Ammann, S. et al. Realistic runtime analysis for quantum simplex computation. Priprint at: https://doi.org/10.48550/arXiv.2311.09995 (2023).
34. Nannicini, G. Fast quantum subroutines for the simplex method. *Oper. Res.* **72**, 763–780 (2024).
35. Kellerer, H., Pferschy, U. & Pisinger, D. *Knapsack Problems* (Springer, 2004).
36. Jooken, J., Leyman, P. & De Causmaecker, P. A new class of hard problem instances for the 0-1 knapsack problem. *Eur. J. Oper. Res.* **301**, 841–854 (2022).
37. Dürr, C. & Høyer, P. A Quantum Algorithm for Finding the Minimum. Preprint at: https://doi.org/10.48550/arXiv.quant-ph/9607014 (1996).
38. Martello, S., Pisinger, D. & Toth, P. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Manage. Sci.* **45**, 414–424 (1999).
39. Bixby, R. E. Solving real-world linear programs: a decade and more of progress. *Oper. Res.* **50**, 3–15 (2002).
40. Pisinger, D. An expanding-core algorithm for the exact 0-1 knapsack problem. *Eur. J. Oper. Res.* **87**, 175–187 (1995).
41. Chakrabarti, S., Minssen, P., Yalovetzky, R. & Pistoia, M. Universal Quantum Speedup for Branch-and-Bound, Branch-and-Cut, and Tree-Search Algorithms. Preprint at: https://doi.org/10.48550/arXiv.2210.03210 (2022).
42. Martiel, S. & Remaud, M. Practical implementation of a quantum backtracking algorithm. Preprint at: https://doi.org/10.48550/arXiv.1908.11291 (2020).
43. Kadowaki, T. & Nishimori, H. Quantum annealing in the transverse Ising model. *Phys. Rev. E* **58**, 5355–5363 (1998).
44. Farhi, E., Goldstone, J., Gutmann, S. & Sipser, M. Quantum Computation by Adiabatic Evolution. Preprint at: https://doi.org/10.48550/arXiv.quant-ph/0001106 (2000).
45. Farhi, E., Goldstone, J. & Gutmann, S. A Quantum Approximate Optimization Algorithm. Preprint at: https://doi.org/10.48550/arXiv.1411.4028 (2014).
46. Hadfield, S. et al. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms* **12**, 34 (2019).
47. Binkowski, L., Koßmann, G., Ziegler, T. & Schwonnek, R. Elementary proof of QAOA convergence. *New J. Phys.* **26**, 073001 (2024).
48. Jooken, J., Leyman, P. & De Causmaecker, P. Knapsack problem instances. https://www.GitHub/dnlfm/knapsack-01-instances (2022).
49. Jooken, J., De Causmaecker, P., Leyman, P., Goedgebeur, J. & Wauters, T. *Monte Carlo Tree Search and Instance Space Analysis for the 0-1 Knapsack Problem*. Ph.D. thesis, Katholieke Universiteit Leuven (2023).
50. Draper, T. G. Addition on a Quantum Computer. Preprint at: https://doi.org/10.48550/arXiv.quant-ph/0008033 (2000).

Acknowledgements

This work was supported by the DFG through SFB 1227(DQ-mat), QuantumFrontiers, theQuantumValley Lower Saxony, the BMBF projects ATIQ and QuBRA, and the BMWK project ProvideQ. Helpful correspondence and discussions with Arne-Christian Voigt, Mark Bennemann, and Timo Ziegler are gratefully acknowledged.

Author contributions

This project was conceived of, and initiated in, discussions of S.W. and T.J.O. The quantum implementation along with the resource estimation procedures were developed by S.W., L.B. and A.I.L. The classical expertise was provided by S.F. and M.P. All authors contributed to writing the paper.

Funding

Open Access funding enabled and organized by Projekt DEAL.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41534-025-01097-8.

Correspondence and requests for materials should be addressed to Sören Wilkening.