Optimization of Robot-Trajectory Planning with Nature-Inspired and Hybrid Quantum Algorithms

Martin J.A. Schuetz,[1,2,3,*] J. Kyle Brubaker,[2] Henry Montagu,[1,2,3] Yannick van Dijk,[4] Johannes Klepsch,[4] Philipp Ross,[4] Andre Luckow,[4] Mauricio G.C. Resende,[5,6] and Helmut G. Katzgraber[1,2,3,6]

[1] *Amazon Quantum Solutions Lab, Seattle, Washington 98170, USA*
[2] *AWS Intelligent and Advanced Compute Technologies, Professional Services, Seattle, Washington 98170, USA*
[3] *AWS Center for Quantum Computing, Pasadena, California 91125, USA*
[4] *BMW Group, Munich, Germany*
[5] *Amazon.com, Inc., Bellevue, Washington 98004, USA*
[6] *University of Washington, Seattle, Washington 98195, USA*

(Received 8 July 2022; revised 20 September 2022; accepted 3 October 2022; published 15 November 2022)

We solve robot-trajectory planning problems at industry-relevant scales. Our end-to-end solution integrates highly versatile random-key algorithms with model stacking and ensemble techniques, as well as path relinking for solution refinement. The core optimization module consists of a biased random-key genetic algorithm. Through a distinct separation of problem-independent and problem-dependent modules, we achieve an efficient problem representation, with a native encoding of constraints. We show that generalizations to alternative algorithmic paradigms such as simulated annealing are straightforward. We provide numerical benchmark results for industry-scale data sets. Our approach is found to consistently outperform greedy baseline results. To assess the capabilities of today's quantum hardware, we complement the classical approach with results obtained on quantum annealing hardware, using `qbsolv` on Amazon Braket. Finally, we show how the latter can be integrated into our larger pipeline, providing a quantum-ready hybrid solution to the problem.

DOI: 10.1103/PhysRevApplied.18.054045

I. INTRODUCTION

The problem of robot motion planning is pervasive across many industry verticals, including (for example) automotive, manufacturing, and logistics. Specifically, in the automotive industry robotic path optimization problems can be found across the value chain in body shops, paint shops, assembly, and logistics, among others [1]. Typically, hundreds of robots operate in a single plant in body and paint shops alone. Paradigmatic examples in modern vehicle manufacturing involve so-called welding jobs, application of adhesives, sealing panel overlaps, or applying paint to the car body. The common goal is to achieve efficient load balancing between the robots, with optimal sequencing of individual robotic tasks within the cycle time of the larger production line.

Another prototypical example involves postwelding processes by which every joint is sealed with special compounds to ensure a car body's water tightness. To this end, polyvinyl chloride (PVC) is commonly applied in a fluid state, thereby sealing the area where different metal parts overlap. The strips of PVC are referred to as *seams*. Post application, the PVC is cured in an oven to provide the required mechanical properties during the vehicle's lifetime. Important vehicle characteristics such as corrosion protection and soundproofing are enhanced by this process. Modern plants usually deploy a fleet of robots to apply the PVC sealant, as schematically depicted in Fig. 1. However, the major part of robot programming is typically carried out by hand, either online or offline. Compared to the famous NP-hard traveling salesman problem, the complexity of identifying optimal robot trajectories is amplified by three major factors. First, an industrial robot arm can have multiple configurations that result in the same location and orientation of the end effector. Furthermore, the PVC is applied with a tool that is equipped with multiple nozzles that allows for application at different angles. A choice must be made regarding which nozzle to use for seams that display easy reachability. Finally, industrial robots are frequently mounted on a linear axis; thus, an optimal location of the robot on the linear axis at which the seam is processed must be determined. The objective of picking and sequencing the robot's trajectories is to find a time-optimal and collision-free production plan.

[*] maschuet@amazon.com

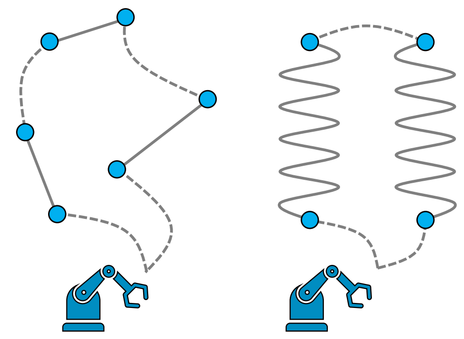

FIG. 1. Schematic illustration of the use case. Robots are programmed to follow certain trajectories along which they apply a PVC sealant along seams. The seams are highlighted by solid lines with two endpoints each, and are not necessarily straight. Dotted lines represent additional motion between seams. Every robot is equipped with multiple tools and tool configurations, to be chosen for every seam. The goal is to identify collision-free trajectories such that all seams get processed within the minimum time span.

Such an optimal production plan may increase throughput, and automation of robot programming reduces the development time of new car bodies.

Quantum computers hold the promise to solve seemingly intractable problems across virtually all disciplines with chemistry and optimization being likely the first medium-term workloads. Specifically, the advent of quantum annealing devices such as the D-Wave Systems Inc. quantum annealers [2–5] has spawned an increased interest in the development of quantum native, heuristic approaches to solve discrete optimization problems. While impressive progress has been made over the last few years, the field is currently still in its infancy, but arguably at a transition point from mere academic research to industrialization. Currently, however, it is still unclear what type of quantum hardware and algorithms will deliver quantum advantage for a practical, real-world problem. Because of this, it is imperative to develop optimization methods that can bridge the gap until scalable quantum hardware is available, but also prepare customers to use specific optimization models that will eventually be able to run on quantum hardware.

The industry use case outlined above has been previously proposed as a potential industry reference benchmark problem for emerging quantum technologies [6–8]. To assess the capabilities of near-term quantum hardware and its potential impact for real-world industry use cases, here we follow a two-pronged approach. On the one hand, we provide and analyze small-scale numerical experiments on quantum annealing hardware (and hybrid extensions thereof), while on the other hand, we design and implement a complementary nature-inspired solution strategy that can integrate quantum computing hardware into its larger framework and provide business value already today. Specifically, we put forward an end-to-end optimization pipeline that extends evolutionary metaheuristics known as biased random-key genetic algorithms [9] towards alternative algorithmic paradigms such as simulated annealing, in conjunction with model stacking and ensemble techniques for solution refinement. For automated hyperparameter tuning, we leverage Bayesian optimization techniques. By design, the resulting hybrid, quantum-ready solution is highly portable and should find applications across a myriad of industry-scale combinatorial optimization problems far beyond the use case studied in this work.

This paper is structured as follows. In Sec. II we review the basic algorithmic concepts underlying our work, with details on biased random-key genetic algorithms, as well as quantum annealing. In Sec. III we then detail our theoretical framework, providing a comprehensive, quantum-ready optimization pipeline for solving robot path problems at industry scales. Section IV describes systematic numerical benchmark experiments. Finally, in Sec. V we draw conclusions and give an outlook on future directions of research.

II. PRELIMINARIES

We start with a brief review of biased random-key genetic algorithms and dual annealing, to set notation and explain our terminology. Furthermore, we give a brief introduction to quantum annealing, as well as the quadratic unconstrained binary optimization (QUBO) formalism.

A. Biased random-key genetic algorithms

Biased random-key genetic algorithms (BRKGAs) [9] represent a (nature-inspired, because genetic) heuristic framework for solving optimization problems. It is a refinement of the random-key genetic algorithm of Bean [10]. While most of the work in the literature has focused on combinatorial optimization problems, BRKGA has also been applied to continuous optimization problems [11]. The BRKGA formalism is based on the idea that a solution to an optimization problem can be encoded as a vector of random keys, i.e., a vector $\mathcal{X}$ in which each entry is a real number, generated at random in the interval (0, 1]. Such a vector $\mathcal{X}$ is mapped to a feasible solution of the optimization problem with the help of a *decoder*, i.e., a deterministic algorithm that takes as input a vector of random keys and returns a feasible solution to the optimization problem, as well as the cost of the solution.

1. BRKGA for traveling salesman and vehicle routing problems

The BRKGA framework is well suited for sequencing-type optimization problems, as relevant for our PVC use case. For example, consider the traveling salesman problem (TSP) where a salesman is required to visit n given cities, each city only once, and do so taking a minimum-length tour. A solution to the TSP is a permutation π of the n cities visited and its cost is

$$c = \ell(\pi_1, \pi_2) + \ell(\pi_2, \pi_3) + \cdots + \ell(\pi_{n-1}, \pi_n) + \ell(\pi_n, \pi_1),$$

where $\ell(i,j)$ is the distance between city i and city j. A possible decoder for the TSP takes the vector of random keys as input and sorts the vector in increasing order of its keys. The indices of the sorted vector make up π, the permutation of the visited cities. As an example, consider a TSP on five cities and let $\mathcal{X} = (0.45, 0.78, 0.15, 0.33, 0.95)$. The sorted vector is $s(\mathcal{X}) = (0.15, 0.33, 0.45, 0.78.0.95)$ and its vector of indices is $\pi = (3, 4, 1, 2, 5)$ having cost

$$c = \ell(3, 4) + \ell(4, 1) + \ell(1, 2) + \ell(2, 5) + \ell(5, 3).$$

Consider now the vehicle routing problem (VRP) where we are given up to p vehicles, a depot (node 0), and n locations $\{1, 2, \ldots, n\}$ that these vehicles must visit, starting and ending at the depot. Each location must be visited by exactly one vehicle and all locations must be visited. A solution to this problem is a set of p permutations $\pi^1, \pi^2, \ldots, \pi^p$ such the $\pi^i \cap \pi^j = \varnothing$ (i.e., no two vehicles visit the same location) for $i = 1, \ldots, p-1$, $j = i+1, \ldots, p$ and $\bigcup_{i=1}^{p} \pi^i = \{1, 2, \ldots, n\}$ (i.e., all locations are visited). In this solution π^i indicates the sequence that vehicle i will take. Suppose that $\pi^1 = \{1, 3, 5\}$ and $\pi^2 = \{4, 2\}$; then vehicle 1 visits locations 1, 3, and 5, in this order, and vehicle 2 visits location 4 and then location 2. Both vehicles start and end their tours at node 0 (the depot). The cost C of this solution is the sum of the costs of the tours of each vehicle, i.e., $C = c^1 + c^2$, where

$$c^1 = \ell(0, 1) + \ell(1, 3) + \ell(3, 5) + \ell(5, 0)$$

and

$$c^2 = \ell(0, 4) + \ell(4, 2) + \ell(2, 0).$$

A possible decoder for the VRP takes as input a vector of $n + v$ random keys, sorts the keys in increasing order of their values, rotates the vector of sorted keys such that the largest of the v keys is last in the array, and then uses the v keys to indicate the division of locations traveled to by each vehicle. For example, consider $n = 5$ and $v = 2$ and consider the vector $\mathcal{X} = (0.45, 0.78, 0.15, 0.33, 0.95, \mathbf{0.25}, \mathbf{0.35})$ of random keys. The first $n = 5$ keys correspond to the locations to be visited by the $v = 2$ vehicles. The last two keys correspond to the two vehicles and are indicated in bold. Sorting the keys in increasing order results in $s(\mathcal{X}) = (0.15, \mathbf{0.25}, 0.33, \mathbf{0.35}, 0.45, 0.78, 0.95)$. The corresponding solution $(3, \mathbf{V_1}, 4, \mathbf{V_2}, 1, 2, 5)$ corresponds to the indices of the sorted random-key vector. For example, 3 is the index of the smallest key, 0.15, while 5 is the index of the largest key, 0.95. Here $\mathbf{V_1}$ and $\mathbf{V_2}$ respectively correspond to the indices of vehicle random keys in the sorted vector. Rotating the elements of the solution vector circularly such that V_2 occupies the last position in the vector, we get $(1, 2, 5, 3, \mathbf{V_1}, 4, \mathbf{V_2})$, which translates into a solution where vehicle V_1 leaves the depot and visits locations 1, 2, 5, 3, and then returns to the depot and vehicle V_2 leaves the depot, visits location 4 and returns to the depot. The cost C of this solution is the sum of the costs of the tours of each vehicle, i.e., $C = c^1 + c^2$, where

$$c^1 = \ell(0, 1) + \ell(1, 2) + \ell(2, 5) + \ell(5, 3) + \ell(3, 0)$$

and

$$c^2 = \ell(0, 4) + \ell(4, 0).$$

2. Anatomy of BRKGA

BRKGA starts with an initial population $\mathcal{P}_0$ of p random-key vectors, each of length N. A decoder is applied to each vector to produce a solution to the problem being solved. Over a number of generations, BRKGA evolves this population until some stopping criterion is satisfied. Populations $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_K$ are generated over K generations. The best solution in the final population is output as the solution of the BRKGA. BRKGA is an elitist algorithm in the sense that it maintains an elite set $\mathcal{E}$ with the best solutions found during the search. The dynamics of the evolutionary process is simple. Population $\mathcal{P}_k$ of each generation is made up of two sets of random vectors: the elite set and the remaining solutions, the nonelite set. To generate population $\mathcal{P}_{k+1}$ from $\mathcal{P}_k$, the elite set of $\mathcal{P}_k$ is copied, without modification to $\mathcal{P}_{k+1}$. This accounts for $p_e = |\mathcal{E}|$ elements. Next, a set $\mathcal{M}$ of mutant solutions (randomly generated random-key vectors) is generated and added to $\mathcal{P}_{k+1}$. This accounts for an additional $p_m = |\mathcal{M}|$ elements. The remaining $p - p_e - p_m$ elements of $\mathcal{P}_{k+1}$ are generated through parameterized uniform crossover [12]. Two parents are selected at random, with replacement: one from the elite set of $\mathcal{P}_k$ and the other from the nonelite set. Denote these parents as the elite parent $\mathcal{X}_a$ and the nonelite parent $\mathcal{X}_b$, respectively. The offspring $\mathcal{X}_c$ is generated as follows. For $i = 1, \ldots, N$, let $\mathcal{X}_c[i] \leftarrow \mathcal{X}_a[i]$ with probability $\Pi > \frac{1}{2}$. Otherwise, with probability $1 - \Pi$, $\mathcal{X}_c[i] \leftarrow \mathcal{X}_b[i]$. Offspring $\mathcal{X}_c$ is added to $\mathcal{P}_{k+1}$. This process is repeated until all $p - p_e - p_m$ offspring are added to $\mathcal{P}_{k+1}$, completing a population of p elements. The p_e random-key vectors with the overall best solutions of $\mathcal{P}_{k+1}$ are placed in the population's elite set, while the remaining vectors are nonelite solutions. A new iteration starts by setting $k \leftarrow k + 1$. For illustration, the evolutionary process underlying BRKGA is illustrated schematically in Fig. 2.

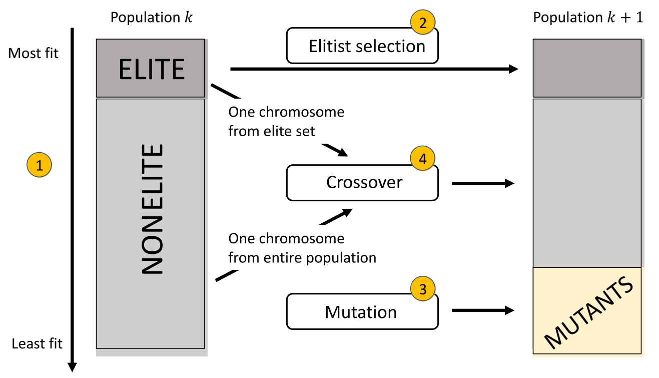

FIG. 2. Schematic illustration of the evolutionary process underlying BRKGA. In step (1) the chromosomes within the current population k are ranked according to their fitness values. In step (2) the elite individuals (those with the highest fitness scores) are copied over to population $k+1$. In step (3), for diversity and to combat local minima, new mutant individuals are randomly generated and added to population $k+1$. In step (4) the remaining portion of population $k+1$ is topped up with offspring generated by a biased crossover that mates elite with nonelite parents.

3. BRKGA in action

BRKGA is a general-purpose optimizer where only the decoder needs to be tailored towards a particular problem. In addition, several hyperparameters need to be specified. These are limited to the length N of the vector of random keys, the size p of the population, the size of the elite set $p_e < p/2$, the size of the set of mutants $p_m \leq p - p_e$, and the probability $\Pi > 1/2$ that the offspring inherits the keys of the elite parent. In addition, a stopping criterion needs to be given. That can be, for example, a maximum number of generations, a maximum number of generations without improvement, a maximum running time, or some other criterion. Several application programming interfaces (APIs) have been proposed for BRKGA [13,14], including some based on C++, PYTHON, JULIA, and JAVA. With these APIs, the user only needs to define a decoder and specify the hyperparameters of the algorithm.

4. Extensions for BRKGA

It should be noted that several extensions have been proposed for BRKGA. Decoding can be done in parallel [15]. Instead of evolving a single population, several populations can be evolved in an island model [16]. Restarts are known to improve the performance of stochastic local search optimization algorithms [17]. The number of generations without improvement can be used to trigger a restart in a BRKGA where the current population is replaced by a population of p vectors of random keys. In BRKGA with restart [18] a maximum number of restarts can be used as a stopping criterion. Instead of mating two parents, mating can be done with multiple parents [19]. Finally, path-relinking strategies can be applied in the space of random keys as a problem-independent intensification operator [14].

B. Dual annealing

Dual annealing (DA) is a stochastic, global (nature-inspired) optimization algorithm. Here, we provide a brief overview of the DA algorithm as used in our extension of the random-key optimizer, described in more depth in Sec. III. We use the DA implementation provided in the SciPy library [20]. This implementation is based on generalized simulated annealing (GSA), which generalizes classical simulated annealing (CSA) and the extended fast simulated annealing (FSA) into one unified algorithm [21,22], coupled with a strategy for applying a local search on accepted locations for further solution refinement. GSA uses a modified Cauchy-Lorentz visiting distribution, whose shape is controlled by the visiting parameter q_v,

$$g_{q_v}(\Delta\mathcal{X}(t)) \propto \frac{[T_{q_v}(t)]^{-D/(3-q_v)}}{[1+(q_v-1)(\Delta\mathcal{X}(t))^2/[T_{q_v}(t)]^{2/(3-q_v)}]^{1/(q_v-1)+(D-1)/2}}, \tag{1}$$

where t is the artificial time (algorithm iteration). This distribution is used to generate a candidate jump distance $\Delta\mathcal{X}(t)$ under temperature T_{q_v}, which is the step from variable $\mathcal{X}(t)$ the algorithm proposes to take. If this proposed step yields an improved cost, it is accepted. If the step does not improve the cost, it may be accepted with acceptance probability

$$p_{q_a} = \min\{1, \max\{0, [1-(1-q_a)\beta\Delta E]^{1/(1-q_a)}\}\}, \tag{2}$$

where ΔE is the change in energy (cost) of the system, q_a is an algorithm hyperparameter, and $\beta \equiv 1/(\kappa T_{q_v}(t))$ refers to the inverse temperature, with Boltzmann constant κ. If the proposed step is accepted, this yields an update step of

$$\mathcal{X}(t) = \mathcal{X}(t-1) + \Delta\mathcal{X}(t); \tag{3}$$

otherwise, $\mathcal{X}(t)$ remains unchanged. The artificial temperature $T_{q_v}(t)$ is decreased according to the annealing schedule

$$T_{q_v}(t) = T_{q_v}(1)\frac{2^{q_v-1}-1}{(1+t)^{q_v-1}-1}, \qquad (4)$$

where $T_{q_v}(1)$ is the starting temperature, with default $T_{q_v}(1) = 5230$. As the algorithm runs through this parameterized annealing schedule, both acceptance probabilities p_{q_a} as well as jump distances $\Delta\mathcal{X}(t)$ decrease over time; this has been shown to yield improved global convergence rates over FSA and CSA [23].

After each GSA temperature step, a local search function is invoked, which in the bounded variable case [as ours is here, i.e., $\mathcal{X}(t) \in [0,1]^n$] defaults to the limited-memory Broyden-Fletcher-Goldfarb-Shanno bound-constrained (L-BFGS-B) algorithm. At each iteration, the L-BFGS-B algorithm runs a line search along the direction of steepest gradient descent around $\mathcal{X}(t)$ while conforming to provided bounds. For more details, see Ref. [24]. Once the local search converges (or exits, i.e., by reaching invocation limits), the found solution $\mathcal{X}(t)$ is used as the starting point for the next step in the GSA algorithm.

This dual annealing process of GSA followed by the L-BFGS-B algorithm runs until convergence, or until the algorithm exits due to maximum iterations, as set by the algorithm's hyperparameter `maxiter`. If the artificial temperature $T_{q_v}(t)$ shrinks to a value smaller than $\mathcal{R} * T_{q_v}(1)$ (with corresponding hyperparameter `restart_temp_ratio`) then the dual annealing process is restarted, with the temperature reset to $T_{q_v}(1)$ and a random (bounded) position is provided for $\mathcal{X}_0$. Note that the algorithm iteration counts are not reset in this case, so the overall algorithm runtime remains tractable.

C. Quantum annealing and the QUBO formalism

Quantum computers are devices that harness quantum phenomena not available to conventional (classical) computers. Today, the two most prominent paradigms for quantum computing involve (universal) circuit-based quantum computers and (special-purpose) quantum annealers [5]. While the former hold the promise of exponential speedups for certain problems, in practice circuit-based devices are extremely challenging to scale up, with current quantum processing units (QPUs) providing about one hundred (*physical*) qubits [5]. Moreover, to fully unlock any exponential speedup, perfect (*logical*) qubits have to be realized, as can be done using quantum error correction, albeit with a large overhead, when encoding one logical (noise-free) qubit in many physical (noisy) qubits. Conversely, quantum annealers are *special-purpose* machines designed to solve certain combinatorial optimization problems belonging to the class of QUBO problems. Since quantum annealers do not have to meet the strict engineering requirements that universal gate-based machines have to meet, already today this technology features about 5000 (physical) analog superconducting qubits.

1. QUBO formalism

Recently, the QUBO framework has emerged as a powerful approach that provides a common modeling framework for a rich variety of NP-hard combinatorial optimization problems [25–28], albeit with the potential for a large variable overhead for some use cases. Prominent examples include the maximum cut problem, the maximum independent set problem, the minimum vertex cover problem, and the traveling salesman problem, among others [25–27]. The cost function for a QUBO problem can be expressed in compact form with the Hamiltonian

$$H_{\mathrm{QUBO}} = \mathbf{x}^{\intercal}Q\mathbf{x} = \sum_{i,j} x_i Q_{ij} x_j, \qquad (5)$$

where $\mathbf{x} = (x_1, x_2, \ldots)$ is a vector of binary decision variables and the QUBO matrix Q is a square matrix that encodes the actual problem to solve. Without loss of generality, the Q matrix can be assumed to be symmetric or in upper triangular form [27]. We have omitted any irrelevant constant terms, as well as any linear terms, as these can always be absorbed into the Q matrix because $x_i^2 = x_i$ for binary variables $x_i \in \{0,1\}$. Problem constraints—which are relevant for many real-world optimization problems—can be accounted for with the help of penalty terms entering the objective function, as detailed in Ref. [27]. Explicit examples will be provided below.

The significance of the QUBO formalism is further illustrated by the close relation to the famous *Ising* model [29], which is known to provide mathematical formulations for many NP-complete and NP-hard problems, including all of Karp's 21 NP-complete problems [25]. As opposed to QUBO problems, Ising problems are described in terms of binary (classical) spin variables $z_i \in \{-1, 1\}$ that can be mapped straightforwardly to their equivalent QUBO form, and vice versa, using $z_i = 2x_i - 1$. The corresponding classical Ising Hamiltonian reads

$$H_{\mathrm{Ising}} = -\sum_{i,j} J_{ij} z_i z_j - \sum_i h_i z_i \qquad (6)$$

with two-body spin-spin interactions $J_{ij} = -Q_{ij}/4$ and local fields h_i (note that a trivial constant has been omitted). If the couplers J_{ij} are chosen from a random distribution, the Ising model given above is also known as a *spin glass*. By definition, both the QUBO and the Ising models are quadratic in the corresponding decision variables. If the original optimization problem involves k-local interactions with $k > 2$, degree reduction schemes have to be involved, at the expense of the aforementioned overhead in terms of the number of variables [5]. In general, one disadvantage of solving problems in a QUBO formalism on quantum annealing hardware lies in the fact that the problem has to be first mapped to a binary representation, then locality has to be reduced to $k \leq 2$ (see below for details).

2. Quantum annealing

Quantum annealing (QA) [30,31] is a metaheuristic for solving combinatorial optimization problems on special-purpose quantum hardware, as well as via software implementations on classical hardware using quantum Monte Carlo [32]. In this approach the solution to the original optimization problem is encoded in the ground state of the so-called problem Hamiltonian $\hat{H}_{\text{problem}}$. Finding the optimal assignment $\mathbf{z}^*$ for the classical Ising model (6) is equivalent to finding the ground state of the corresponding problem Hamiltonian, where we have promoted the classical spins $\{z_i\}$ to quantum spin operators $\{\hat{\sigma}_i^z\}$, also known as Pauli matrices, thus describing a collection of interacting qubits. To (approximately) find the *classical* solution $\{z_i^*\}$, quantum annealing devices then undergo the following protocol. Start with the algorithm by initializing the system in some easy-to-prepare ground state of an initial Hamiltonian $\hat{H}_{\text{easy}}$, which is chosen to not commute with $\hat{H}_{\text{problem}}$. Following the adiabatic approximation [5], the system is then slowly annealed towards the so-called problem Hamiltonian $\hat{H}_{\text{problem}}$, whose ground state encodes the (hard-to-prepare) solution of the original optimization problem. This is commonly done in terms of the annealing parameter τ, defined as $\tau = t/T_A \in [0,1]$, where t is the physical wall-clock time and T_A is the annealing time. In the course of this anneal, ideally, the probability to find a given classical configuration converges to a distribution that is strongly peaked around the ground state of H_{Ising}. Overall, the protocol is captured by the time-dependent Hamiltonian

$$\hat{H}(\tau) = A(\tau)\hat{H}_{\text{easy}} + B(\tau)\hat{H}_{\text{problem}}, \tag{7}$$

where the functions $A(\tau), B(\tau)$ describe the annealing schedule, with $A(0)/B(0) \gg 1$ and $A(1)/B(1) \ll 1$. For example, a simple, linear annealing schedule is given by $A(\tau) = 1 - \tau$ and $B(\tau) = \tau$. Because of manufacturing constraints, current experimental devices can only account for 2-local interactions, with a cost function described by

$$\hat{H}_{\text{problem}} = -\sum_{i,j} J_{ij}\hat{\sigma}_i^z\hat{\sigma}_j^z - \sum_i h_i\hat{\sigma}_i^z, \tag{8}$$

while the initial Hamiltonian is typically chosen as

$$\hat{H}_{\text{easy}} = -\sum_i \hat{\sigma}_i^x, \tag{9}$$

which is a transverse-field driving Hamiltonian responsible for quantum tunneling between the classical states making up the computational basis states. Because the final Hamiltonian (8) only involves commuting operators $\{\hat{\sigma}_i^z\}$, the final solution $\{z_i^*\}$ can be read out as the state of the individual qubits via a measurement in the computational basis.

3. Embedding

Solving an optimization problem of QUBO form on QA hardware, however, frequently involves one more step, typically referred to as *embedding* [33]. Because of manufacturing constraints, today's quantum annealers based on superconducting technology only come with limited connectivity, i.e., not every qubit is physically connected to every other qubit. In fact, typically, the on-chip matrix J_{ij} is sparse. If, however, the problem's required logical connectivity does not match that of the underlying hardware, one can effectively replicate the former using an embedding strategy by which several *physical* qubits are combined into one *logical* qubit. The standard approach to do so is called *minor embedding* that provides a mapping from a (logical) graph to a subgraph of another (hardware) graph. One can then solve high-connectivity problems directly on the sparsely connected chip, by sacrificing physical qubits accordingly to the connectivity of the problem, typically introducing a considerable overhead with multiple physical qubits making up one logical variable. This limitation makes the problems subsequently harder to solve than in their native formulation [34]. Specifically, in the extreme case of a fully connected graph (as relevant for the traveling salesman problem) only approximately 64 *logical* spin variables can be embedded within the D-Wave 2000Q quantum annealer that nominally features about 2000 *physical* qubits. Another example, circuit fault diagnosis, is analyzed in detail in Ref. [35] and further highlights some of these limitations. Finally, when including constrained problems, the coupler distributions tend to broaden, which, in turn, results in an additional disadvantage due to the limited precision of the analog device.

4. QA workflow

Today, quantum annealing devices as provided by D-Wave Systems Inc. can be conveniently accessed through the cloud. In Fig. 3 we detail the end-to-end workflow for solving some combinatorial optimization problem on such a quantum annealer from a practitioner's point of view. Below, we put this workflow into practice when solving small instances of our use case on D-Wave, as available on Amazon Braket, and hybrid extensions thereof.

III. THEORETICAL FRAMEWORK

In this section we discuss in detail the theoretical framework underlying our work, as outlined in Fig. 4. We first define notation, introducing the concept of an abstract, composite *node* that encapsulates information about the

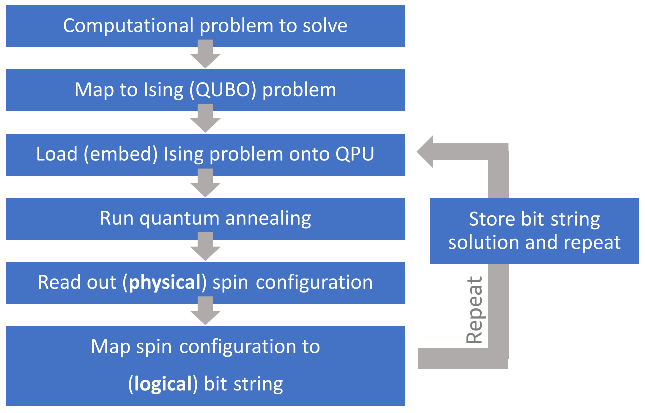
Computational problem to solve
Map to Ising (QUBO) problem
Load (embed) Ising problem onto QPU
Run quantum annealing
Read out (**physical**) spin configuration
Map spin configuration to (**logical**) bit string
Store bit string solution and repeat
Repeat

FIG. 3. Flow chart illustrating the end-to-end workflow for solving a combinatorial optimization problem on a quantum annealer. First, the problem has to be cast as a QUBO (or, equivalently, Ising) Hamiltonian. This abstract QUBO problem is then mapped onto the physical QPU, typically at the expense of an enlarged number of variables (given the sparse connectivity of the underlying quantum chip). Next, quantum annealing is used to find a high-quality variable configuration. This solution is mapped back to a bit string (of logical variables) corresponding to a solution of the original optimization problem. Given the probabilistic nature of this process, it is typically repeated multiple times, followed by a statistical analysis. The goal is to find a configuration of variables that (approximately) minimizes the objective function. Further details are provided in the main text.

seam number together with other relevant degrees of freedom (such as tool configuration or position of the robot). Next we detail our classical as well as quantum native solution strategies to the PVC use case. Specifically, we show how BRKGA can be applied to the PVC use case, by proposing an efficient decoder design that natively accounts for the problem's constraints. Finally, we detail how this use case can be described within the quantum-ready QUBO framework.

A composite *node* can be viewed as a generalization of a *city* in the canonical TSP. In analogy of the TSP, the goal is to identify an optimal sequence of nodes, with a node encoding not only spatial information, but also other categorical features relevant to the use case at hand. Specifically, in our setup we define a node as a quintuple of the form

$$\text{node} = [s, d, t, c, p]. \tag{10}$$

Generalizations to other problems are straightforward. Here, a node encapsulates information about the seam index $s = 1, \ldots, n_{\text{seams}}$, the direction $d = 0, 1$ by which a given seam is sealed, the tool $t = 1, \ldots, n_{\text{tools}}$, and tool configuration $c = 1, \ldots, n_{\text{config}}$ used, as well as the (discretized) linear-axis position $p = 1, \ldots, n_{\text{position}}$. This definition of a generalized, composite node accounts for

TABLE I. Example data set (cost matrix) for illustration. The problem is specified in terms of cost values (in seconds) for pairs of nodes, with every node described by a tuple $[s, d, t, c, p]$ that captures the seam index s, direction d, tool t, tool configuration c, and robot position p. Further details are provided in the main text.

Node (from)					Node (to)					
s	d	t	c	p	s	d	t	c	p	Cost w (s)
0	0	0	0	0	18	1	1	1	1	0.877
11	2	1	0	1	12	1	2	0	1	0.473
11	2	1	0	1	12	0	3	0	0	0.541
32	2	1	2	1	25	2	1	2	1	0.558
⋮					⋮					⋮

the facts that (i) any seam can be sealed in one of two directions, (ii) a robot can seal a given seam using one of several tools, (iii) which (at the same time) can be employed in different configurations, and (iv) a robot can take one of several positions along a fixed rail. All coordinates can be described by integer values. The problem is then specified in terms of cost values $w_{\text{node}_i}^{\text{node}_j}$ (in seconds) for the robot to move from one of the endpoints of node$_i$ to one of the endpoints of node$_j$, including both the cost associated with applying PVC to node$_i$ as well as proceeding in idle mode to node$_j$. As illustrated in Fig. 1, every seam has two endpoints, i.e., a degree of freedom captured by the binary direction variable $d = 0, 1$. For illustration, a sample data set is shown in Table I. For industry-relevant problem instances, such a data set has roughly one million rows, only providing preselected, *feasible* connections, as (in practice) many node pairs represent infeasible robot routes because of obstacles. Finally, we note that each robot has a home (or base) position from which it starts its operation, and where its tour comes to an end; this home position is associated with node $[0, 0, 0, 0, 0]$. More details on the generation of real-world data sets will be given in Sec. IV.

A. Robot motion planning with nature-inspired algorithms

We now discuss our end-to-end optimization pipeline for robot motion planning, as schematically depicted in Fig. 4. We first detail the core optimization routine, dubbed random-key optimizer (RKO), with a focus on the use-case-specific decoder design, before discussing potential upstream and downstream extensions for further solution refinement. Specifically, with the RKO concept, we introduce a generalization of the BRKGA formalism and its distinct separation of problem-independent and problem-dependent modules towards alternative optimization paradigms such as simulated annealing. More details are provided below.

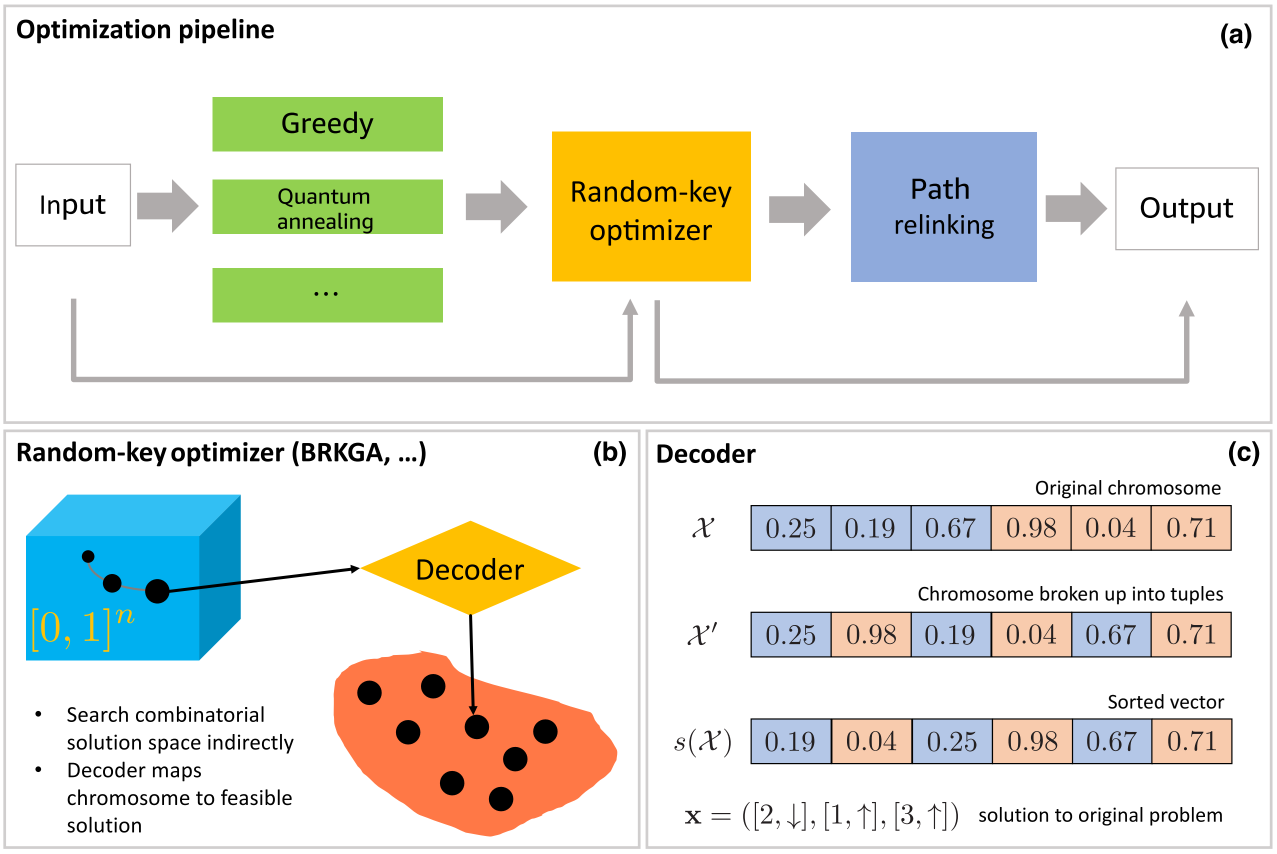
Optimization pipeline
(a)
Input
Greedy
Quantum annealing
...
Random-key optimizer
Path relinking
Output
Random-key optimizer (BRKGA, ...)
(b)
$[0,1]^n$
Decoder
Search combinatorial solution space indirectly
Decoder maps chromosome to feasible solution
Decoder
(c)
Original chromosome
$\mathcal{X}$ 0.25 0.19 0.67 0.98 0.04 0.71
Chromosome broken up into tuples
$\mathcal{X}'$ 0.25 0.98 0.19 0.04 0.67 0.71
Sorted vector
$s(\mathcal{X})$ 0.19 0.04 0.25 0.98 0.67 0.71
$\mathbf{x} = ([2,\downarrow],[1,\uparrow],[3,\uparrow])$ solution to original problem

FIG. 4. Schematic illustration of our approach. (a) Flow chart of our end-to-end optimization pipeline. The core routine takes the problem input, here specified in terms of cost values associated with pairs of nodes, and feeds this input into the random-key optimizer (RKO). The latter heuristically searches for an optimized tour of composite nodes, which represents the pipeline's output. This core routine can be extended with additional upstream and downstream modules for further solution refinement. Upstream solutions provided by alternative algorithms (such as greedy algorithms, quantum annealing, etc.) can be used to warmstart the RKO. Akin to ensemble techniques routinely used in machine learning, a pool of different solutions may help identify high-quality solutions, adding high-quality diversity into the initial population of the RKO module. Downstream, further solution refinement can be achieved through path-relinking techniques, by forming superpositions of high-quality solution candidates. (b) Schematic illustration of the RKO. The key characteristic of the RKO is a clear separation between problem-independent modules (as illustrated by the blue hypercube that hosts the chromosome $\mathcal{X}$) and the problem-dependent, deterministic decoder that maps $\mathcal{X}$ to a solution of the original problem with associated cost (or fitness) value. By design, our decoder ensures that problem constraints (e.g., every node has to be visited exactly once) are satisfied. In BRKGA the trajectory of chromosome $\mathcal{X}$ is set by evolutionary principles, but generalizations to alternative algorithmic paradigms such as simulated annealing are straightforward. (c) Example illustration of the decoding of chromosome $\mathcal{X}$, made of random keys in $(0,1]$, into a solution to the original combinatorial optimization problem. We consider a sequencing problem paired with a binary decision variable (such as the binary direction variable $d = \uparrow, \downarrow$) for $n = 3$ composite nodes. The first block of the chromosome (highlighted in blue) encodes the solution to the sequencing problem, while the second block (highlighted in red) encodes the additional binary variable, thus representing a minimal example for the concept of a *composite* node. The chromosome can be broken up into n tuples, one encoding a single node each. The decoder then performs simple sorting according to the first entry in the tuple, yielding the sorted vector $s(\mathcal{X})$. Finally, the solution to the original problem $\mathbf{x}$ is given by the indices of this sorting routine paired with a simple threshold routine applied to the tuples' second entry. Here, the threshold is set at 0.5. Further details are provided in the main text.

1. Core pipeline: decoder design

We now detail an example decoder design, as used in our numerical experiments. The problem input is given in terms of a generalized cost matrix as displayed in Table I. Similar to the TSP, our goal is to identify a minimum-cost tour (as a sequence of *nodes*) that visits n given *seams*, each seam only once, while specifying the additional degrees of freedom making up a composite *node*. As illustrated in Fig. 4(b), the *decoder* plays an integral part in our random-key approach. While the problem *encoding* is specified by the evolutionary part underlying BRKGA, *decoding* is controlled through the design of the decoder. Here, the decoder is designed as follows (see Appendix A for implementation details in PYTHON code). The decoder takes a vector $\mathcal{X}$ of $N = \mathcal{D}\, n_{\text{seams}}$ random keys as input, sorts the keys associated with the seam numbers s in increasing order of their values, and applies simple thresholding logic to the remainder of keys; see Fig. 4(c) for an illustration. In our case the number of features $\mathcal{D}$ is $\mathcal{D} = 5$. Similar to the TSP example outlined above, the indices of the sorted vector component make up π, the permutation of the visited seams; see the blue block in Fig. 4(c). As opposed to the TSP, however, we have to assign discrete values for the remaining node degrees of freedom as well, as shown in the red block in Fig. 4(c). For example, if the corresponding original variable is binary, as is the case in Fig. 4(c), the thresholding logic reduces to $\text{int}(\mathcal{X}_i) \in \{0, 1\}$, but generalizations to variables with larger cardinality are straightforward. For example, if a larger cardinality is assumed for the original variable $\mathcal{Y}_i$, say $\mathcal{Y}_i \in \{1, 2, \ldots, C\}$, then $\mathcal{Y}_i = k$ if $\mathcal{X}_i \in ((k-1)/C, k/C]$ for $k = 1, 2, \ldots, C$. Note that our mathematical representation by design generates feasible routes only where every seam is visited exactly once, while only scaling linearly with the number of seams n_{seams}, and with a prefactor set by the number of degrees of freedom. While the original cost matrix as displayed in Table I features feasible connections only, the decoder may suggest infeasible moves that have been preselected from the original data set. By padding the cost matrix with prohibitively large cost values for these types of moves, over the course of the evolution the algorithm will learn to steer away from these bad-fitness solutions. As detailed in Sec. IV numerically, we find that our solution always arrives at feasible, low-cost tours that include feasible moves only.

2. Algorithmic generalizations

In the common BRKGA framework the trajectory of every chromosome $\mathcal{X}$ in the abstract half-open hypercube of dimension $(0, 1]^n$ is governed by evolutionary principles, as detailed in Sec. II. However, alternative algorithmic paradigms such as simulated annealing (SA) [36] and related methods can be readily used as well, all within our random-key formalism. That is because, for any chromosome $\mathcal{X}$, the decoder does not only provide the decoded solution $s(\mathcal{X})$ but also the fitness (or cost) value $f(\mathcal{X})$, in our case defined as the total cost of the tour. Black-box query access to $f(\mathcal{X})$, however, is sufficient for optimization routines such as SA to perform an update on the solution candidate $\mathcal{X}$ and continue with training till some (algorithm-specific) stopping criterion is fulfilled. To illustrate this point, we have performed numerical experiments based on the dual annealing algorithm [22]. As detailed in Sec. II, this stochastic approach combines classical SA with local search strategies for further solution refinement. We refer to this annealing-based extension of the random-key formalism as RKO DA. Numerical results and more details are presented in Sec. IV.

3. Warmstarting

The core optimization routine outlined above can be extended with additional upstream logic. Specifically, in analogy to model-stacking techniques commonly used in machine-learning pipelines, solutions provided by alternative algorithms (such as linear programming, greedy algorithms, quantum annealing, etc.) can be used to warmstart the RKO, as opposed to coldstarts with a random initial population $\mathcal{P}_0$. Similar to standard ensemble techniques, the output of several optimizers may be used to seed the input for RKO, thereby leveraging information learned by these while injecting diverse quality into the initial solution pool $\mathcal{P}_0$. By design, this strategy can only improve upon the solutions already found, as elite solutions are not dismissed and just propagate from one population to the next in the course of the evolutionary process. The technical challenge is to invert the decoder with its inherent many-to-one mapping. To this end, we propose the following randomized heuristic. Consider a given permutation π such as $\pi = (4, 2, 3, 1)$, with $n = 4$. Our goal is then to design an algorithm that produces a random key $\mathcal{X}$ that (when passed to the decoder) is decoded to the permutation π. To this end, we chop up the interval $(0, 1]$ into evenly spaced chunks of size $\Delta = 1/n = 0.25$, with centers $\bar{\mathcal{X}}_i$ at 0.125, 0.375, 0.625, and 0.875. We then loop through the input sequence π and assign these center values to appropriate positions in $\mathcal{X}$, as $\mathcal{X} = (\cdot, \cdot, \cdot, \cdot) \rightarrow (\cdot, \cdot, \cdot, 0.125) \rightarrow (\cdot, 0.375, \cdot, 0.125) \rightarrow \cdots \rightarrow (0.875, 0.375, 0.625, 0.125)$. When sorting this key, we obtain the desired sequence of indices given by $\pi = (4, 2, 3, 1)$. Finally, we randomize this deterministic protocol by adding uniform noise $\delta_i \in (\bar{\mathcal{X}}_i - \Delta/2, \bar{\mathcal{X}}_i + \Delta/2)$ to each element in $\mathcal{X}$, thus providing a randomized chromosome such as $\mathcal{X} = (0.93, 0.31, 0.67, 0.08)$. By repeating the last step m times, one can generate a pool of m warm chromosomes. For the remaining categorical features d, t, c, p, it is straightforward to design a similar randomized protocol. For example, for the binary feature d, we generate a random number in $(0, 0.5]$ if $d = 0$, and a random number in $(0.5, 1]$ if $d = 1$.

4. Path relinking

Our core pipeline can be further refined with path-relinking (PR) strategies for potential solution refinement. Several PR strategies are known, some of them operating in the space of random keys (also known as implicit PR [14]) for problem-independent intensification, and some of them operating in the decoded solution space. The common theme underlying all PR approaches is to search for high-quality solutions in a space spanned by a given pool of elite solutions, by exploring trajectories connecting elite solutions [37,38]; one or more paths in the search space graph connecting these solutions are explored in the search for better solutions. In addition to these existing approaches, here we propose a simple physics-inspired PR strategy that can be applied post-training. Consider two high-quality chromosomes labeled as $\mathcal{X}_1$ and $\mathcal{X}_2$. We can then heuristically search for better solutions with the hybrid (superposition) ansatz

$$\mathcal{X}(\alpha) = (1-\alpha)\mathcal{X}_1 + \alpha\mathcal{X}_2 \tag{11}$$

with $\alpha \in [0, 1]$. We then scan the parameter α and query the corresponding fitness by invoking the decoder (without having to run the RKO routine again). For $\alpha = 0$ and $\alpha = 1$, we recover the existing chromosomes $\mathcal{X}_1$ and $\mathcal{X}_2$, respectively, but better solutions may be found along the trajectory sampled with the hybridization parameter α.

5. Restarts

Finally, restart strategies as described in Sec. II can be readily integrated into our larger optimization pipeline. Numerical experiments including restarts are detailed in Sec. IV.

B. Robot motion planning with quantum native algorithms

We now detail a quantum native QUBO formulation for our industry use case, followed by resource estimates for the number of logical qubits required to solve this use case at industry-relevant scales.

1. QUBO representation

We introduce binary (one-hot encoded) variables, setting $x_{\text{node}}^{[\tau]} = 1$ if we visit node $= [s, d, t, c, p]$ at position $\tau = 1, \ldots, n_{\text{seams}}$ of the tour, and $x_{\text{node}}^{[\tau]} = 0$ otherwise. Following the QUBO formulation for the canonical TSP problem [25], we can then describe the goal of finding a minimal-time tour with the quadratic Hamiltonian

$$H_{\text{cost}} = \sum_{\tau=1}^{n_{\text{seams}}} \sum_{\text{node}} \sum_{\text{node}'} w_{\text{node}}^{\text{node}'} x_{\text{node}}^{[\tau]} x_{\text{node}'}^{[\tau+1]} \tag{12}$$

with $w_{\text{node}}^{\text{node}'}$ denoting the cost to go from node to node$'$. Here, the product $x_{\text{node}}^{[\tau]} x_{\text{node}'}^{[\tau+1]} = 1$ if and only if node is at position τ in the cycle and node$'$ is visited right after at position $\tau + 1$. In that case we add the corresponding distance $w_{\text{node}}^{\text{node}'}$ to our objective function that we would like to minimize. Overall, we sum all costs of the distances between successive nodes. Next, we need to enforce the validity of the solution through additional penalty terms, i.e., we need to account for the following constraints. First, we should have exactly one node assigned to every time step in the cycle. Mathematically, this constraint can be written as

$$\sum_{s,d,t,c,p} x_{[s,d,t,c,p]}^{[\tau]} = 1 \quad \text{for all } \tau = 1, \ldots, n_{\text{seams}}. \tag{13}$$

Second, every seam should be visited once and only once (in some combination of the remaining features). Note that we do not have to visit every potential node. This constraint is mathematically captured by

$$\sum_{\tau} \sum_{d,t,c,p} x_{[s,d,t,c,p]}^{[\tau]} = 1 \quad \text{for all } s = 1, \ldots, n_{\text{seams}}. \tag{14}$$

As detailed in Ref. [27] within the QUBO formalism, we capture these constraints through additional penalty terms given by

$$H_{\text{time}} = P \sum_{\tau=1}^{n_{\text{seams}}} \left[\sum_{\text{node}} x_{\text{node}}^{[\tau]} - 1 \right]^2, \tag{15}$$

$$H_{\text{complete}} = P \sum_{s=1}^{n_{\text{seams}}} \left[\sum_{\tau} \sum_{d,t,c,p} x_{[s,d,t,c,p]}^{[\tau]} - 1 \right]^2, \tag{16}$$

with penalty parameter $P > 0$ enforcing the constraints. Note that the numerical value for P can be optimized in an outer loop. Finally, the Hamiltonian describing the robot motion (RM) use case (H_{RM}) then reads

$$H_{\text{RM}} = H_{\text{cost}} + H_{\text{time}} + H_{\text{complete}}. \tag{17}$$

Because the Hamiltonian H_{RM} is quadratic in the binary decision variables $\{x_{\text{node}}^{[\tau]}\}$, it falls into the broader class of QUBO problems, which is amenable to quantum native solution strategies, for example in the form of quantum annealing [5], in addition to traditional classical solvers such as simulated annealing or tabu search.

2. Resource estimation

We complete this analysis with a rough estimate for the number of (logical) qubits n_{qubits} required to implement this QUBO formulation for industry-relevant scales. With the number of time steps for the Hamiltonian cycle given by n_{seams} we obtain

$$n_{\text{qubits}} = 2n_{\text{seams}}^2 n_{\text{tools}} n_{\text{config}} n_{\text{position}}. \tag{18}$$

Taking numbers from a realistic industry-scale case we use $n_{\text{seams}} \sim 50$, $n_{\text{tools}} \sim 3$, $n_{\text{config}} \sim 10$, and $n_{\text{position}} \sim 3$; we then find that $n_{\text{qubits}} \sim 5 \times 10^5$. Furthermore, given the quadratic overhead for embedding an all-to-all connected graph onto the sparse Chimera architecture [35], we can estimate the required number of *physical* qubits N_{qubits} to be as large as

$$N_{\text{qubits}} \sim 10^{11}. \tag{19}$$

This number is much larger than the number of qubits available today and in the foreseeable future, and higher connectivity between the physical qubits will be needed to reduce our requirements for N_{qubits}. Therefore, in our numerical experiments we utilize hybrid (quantum classical) solvers that heuristically decompose the original QUBO problem into smaller subproblems that are compatible with today's hardware. These subproblems are then solved individually on a quantum annealing backend, and a global bit string is recovered from the pool of individual, small-scale solutions by stitching together individual bit strings.

Finally, for illustration purposes, let us also compare the size of the combinatorial search space any QUBO-based approach is exposed to, as compared to the native search space underlying the random-key formalism. Disregarding (for simplicity) all complementary categorical features for now, the size of the search space for the QUBO formalism amounts to $2^{n_{\text{seams}}^2}$, while a native encoding has to search through the space of permutations of size $n_{\text{seams}}!$. This means that, for $n_{\text{seams}} \sim 50$, the latter amounts to about 10^{64}, while the QUBO search space is many orders of magnitude larger, with $2^{50^2} \sim 10^{752}$ possible solution candidates, thus demonstrating the benefits of an efficient, native encoding for sequencing-type problems as relevant here.

IV. NUMERICAL EXPERIMENTS AND BENCHMARKS

We now turn to systematic numerical experiments for industry-relevant data sets. We first describe the generation of the data sets, and then compare results achieved with random-key algorithms to baseline results using greedy methods. Our implementation of BRKGA is based on code originating from Ref. [14]. We also provide results achieved with SA applied within the QUBO modeling framework (referred to as QUBO SA). To assess the capabilities of today's quantum hardware, we complement these classical approaches with quantum native solution strategies, including results obtained on quantum annealing hardware within a hybrid quantum classical algorithm, using `qbsolv` on Amazon Braket (referred to as QUBO QBSolv) [39].

A. Data sets

All data sets are generated by BMW Group using custom logic that is implemented in the Robot Operating System (ROS) framework [40]. To be able to calculate the relevant cost values (measured in seconds) to move between nodes and build the cost matrix W, the physical robot cell is modeled in ROS. This includes loading and positioning of the robot model, importing static collision objects, and parsing the robotic objectives. The generation of the data set is then done in two steps. First, a reachability analysis is carried out to inspect the different possibilities of applying sealant to a given seam. This includes the choice of the nozzle to use, the robot's joint configuration, and the position of the robot on the linear axis. The latter is discretized to provide a finite number of possible linear axis positions to be searched, with n_{position} left as a free parameter in the framework. Second, the motion planning is carried out to obtain the trajectories for all possibilities to move from one seam to all possibilities of processing any other seam. Collision avoidance and time parameterization are included in this process. To this end, we adopt the RRT* motion planning algorithm [41]. If the motion planning does not succeed, no (weighted) edge is entered in the motion graph. The runtime to generate the largest data set presented in this paper is approximately 2.5 d. These calculations are performed with 70 threads on an Intel Xeon Gold 6154 CPU, running at 3.00 GHz. The largest data set considered contains $n_{\text{seams}} = 71$ seams, including the home position. For our scaling and benchmarking analysis (as shown in Fig. 5 below), we have implemented a random downsampling strategy. By randomly removing seams from the original data set we have generated synthetic data sets with variable size, ranging from $n_{\text{seams}} = 10$ to $n_{\text{seams}} = 70$ in increments of five seams, with ten distinct downsamples per system size. For reference, within the QUBO formalism, the smallest instance with $n_{\text{seams}} = 10$ already requires approximately 10^4 binary variables, thus representing a large QUBO problem to be solved.

B. Benchmarking results

First we provide results for two selected, large-scale, industry-relevant benchmark data sets, referred to as benchmark L and benchmark XL with $n_{\text{seams}} = 52$ and $n_{\text{seams}} = 71$, respectively. While benchmark XL represents the largest available problem instance, benchmark L, while smaller, represents a particularly hard instance because it features many obstructions to the robot's path (resulting in missing entries in the motion graph). We compare three different algorithmic strategies on these benchmark instances, including BRKGA and RKO DA (both based on the random-key formalism, but following different optimization paradigms), as well as a greedy baseline. Here we do not provide results for any QUBO-based solution strategies, because these instances are out of reach for QUBO solvers, with approximately 6×10^5 and 10^6 binary variables, respectively. Note that numerical results based on the QUBO formalism will be provided below. Hyperparameter optimization for the BRKGA and RKO-DA solvers is done as outlined in Ref. [42] using Bayesian

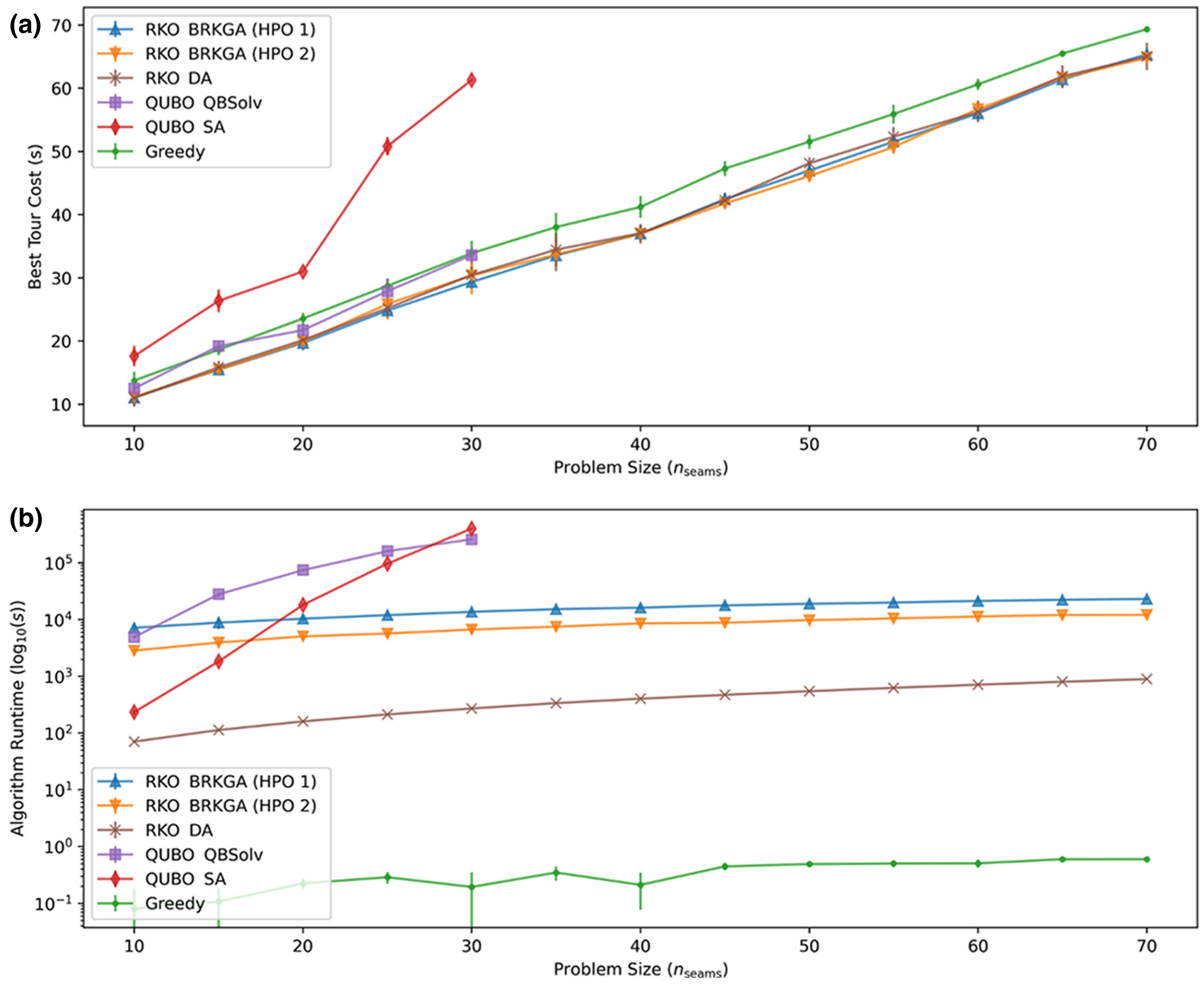

FIG. 5. Numerical results for systems with up to $n_{\text{seams}} = 70$ seams. Other dimensions are fixed, with $n_d = 2$, $n_{\text{tools}} = 3$, $n_{\text{config}} = 9$, $n_{\text{position}} = 4$. For $n_{\text{seams}} = 20$, the QUBO size as measured by the required number of binary variables amounts to about 10^5, posing limits on the accessible problem size because of memory restrictions. (a) Solution quality as measured by the total cost (i.e., duration) of the best tour found (in seconds). All results have been averaged over ten samples per system size, specified by the number of seams. We compare results achieved with BRKGA for two sets of hyperparameters (as shown with the blue upward triangles [HPO 1] and orange downward triangles [HPO 2]) and for RKO-DA (brown crosses) with one set of hyperparameters, with a simple greedy heuristic serving as the baseline (green circles). For instances with $n_{\text{seams}} \lesssim 20$, we provide results based on the QUBO formalism for both classical simulated annealing (QUBO SA, red diamonds), as well as a hybrid (quantum classical) decomposing solver `qbsolv` (QUBO QBSolv, purple squares). (b) Algorithm runtime as a function of the number of seams n_{seams}. The greedy baseline algorithm is extremely fast with its runtime showing practically no discernible dependence on the system size in the parameter regime tested here. Our implementation of BRKGA displays a mild, linear scaling with the absolute numbers depending on hyperparameters such as population size. Still, the largest problems with $n_{\text{seams}} \sim 70$ seams are solved within a few hours. The RKO-DA implementation is found to be about an order of magnitude faster than BRKGA. The QUBO-based approaches display comparatively long runtimes for sufficiently large system sizes. Further details are given in the text.

optimization techniques. The greedy algorithm is run 10^4 times, each time with a different random starting configuration, thereby effectively eliminating any dependence on the random seed. We have observed convergence for the best greedy result after typically about 10^3 shots, and only the best solutions found are reported. Our results are displayed in Table II. We find that both the BRKGA and RKO-DA strategies outperform the greedy baseline. Specifically, BRKGA yields an improvement of 3.24 s (8.74%) on benchmark L and 0.90 s (1.36%) on benchmark XL, while RKO DA yields an even larger improvement of 3.75 s (10.12%) on benchmark L and 2.39 s (3.62%) on benchmark XL. These improvements can directly translate into cost savings, increased production volumes or both.

TABLE II. Numerical results for two industry-relevant benchmark data sets, referred to as benchmark L and benchmark XL. For reference, with QUBO size we report the approximate number of binary variables [cf. Eq. (18)] to describe this instance within the QUBO formalism. We report cost values achieved with three different algorithmic strategies (greedy, BRKGA, and RKO DA). The greedy algorithm has been run 10^4 times, and the best (lowest cost) solution is reported in the table. Best results across algorithms are marked in bold. The last two columns specify the absolute and relative improvements of the best solution over the greedy baseline strategy. Further details are provided in the main text.

Data set	Number of seams	QUBO size	Greedy	BRKGA	RKO DA	Absolute Δ (s)	Relative Δ (%)
Benchmark L	52	6×10^5	37.05	33.81	**33.30**	3.75	10.12
Benchmark XL	71	1×10^6	65.99	65.09	**63.60**	2.39	3.62

C. Scaling results

To complement our benchmark results, we have performed systematic experiments on data sets with variable size, ranging from $n_{\text{seams}} = 10$ to $n_{\text{seams}} = 70$ in increments of five seams, with ten samples per system size. Our results are displayed in Fig. 5, with every curve referring to one fixed set of hyperparameters (such as population size p, mutant percentage p_m, etc. in the case of BRKGA). Further details regarding hyperparameters can be found in Appendix B. We also report numerical results based on the quantum native QUBO formalism, using both classical simulated annealing, as well as a hybrid (quantum classical) decomposing solver `qbsolv`. However, for $n_{\text{seams}} = 20$, the QUBO size as measured by the required number of binary variables already amounts to about 10^5, posing limits on the accessible problem size because of memory restrictions. We compare results achieved with BRKGA for two sets of hyperparameters, and for RKO DA with one set of hyperparameters, and find that these consistently outperform greedy baseline results, providing about a 10% improvement for real-world systems with $n_{\text{seams}} \sim 50$. Note that BRKGA (HPO 2) is run with ten shots for problem sizes 45, 50, 55, and 70 to reduce variance observed when using fixed hyperparameters. Performance of the RKO-DA solver is found to be competitive with BRKGA throughout the range of problem sizes. The QUBO-SA solver is found to be the least competitive, and the least scalable, unable to solve problems beyond $n_{\text{seams}} \approx 20$. Finally, the QUBO-QBSolv approach performs on par with the greedy algorithm, albeit at much longer runtimes, but is unable to scale beyond $n_{\text{seams}} \approx 30$.

Next, we report on the average runtimes of each solver as a function of problem size; see Fig. 5(b). Given its simplicity, the greedy algorithm is found to be the fastest solver, taking under a second (about 0.60 s) to solve the largest problem instances with $n_{\text{seams}} = 70$, after loading in the requisite data. Our implementation of the random-key optimizer RKO DA can solve the largest problem instances within about 15 min, and exhibits a benign, linear runtime scaling with problem size. Our implementation of BRKGA displays a similar scaling, but typically takes a few hours to complete, with average runtimes ranging from about 0.78 h (HPO 2, $n_{\text{seams}} = 10$) to about 6.40 h (HPO 1, $n_{\text{seams}} = 70$). The observed difference in runtimes between BRKGA (HPO 1) and BRKGA (HPO 2) can be largely attributed to the different population sizes (with $p = 9918$ and $p = 7978$ for HPO 1 and HPO 2, respectively) and patience values (with patience = 52 and patience = 66 for HPO 1 and HPO 2, respectively). The observed speedup from BRKGA to RKO DA, both based on the random-key formalism, can be attributed to the internal anatomy of the respective algorithm: while BRKGA evolves entire populations of (in our case typically $p \sim 10^4$) interacting chromosomes, RKO DA tracks only one single chromosome $\mathcal{X}$ throughout its algorithmic evolution, yielding about an order-of-magnitude speedup across the range of problem sizes studied here. Finally, as compared to the heuristics described above, the QUBO-based approaches display unfavorable runtimes, either because of orders-of-magnitude longer runtimes across the range of accessible problem sizes (QUBO QBSolv) or because of unfavorable runtime scaling (QUBO SA), further corroborating the advantage of our native solution strategies as compared to quantum native QUBO formulations.

D. Time-to-target results

We complete our numerical benchmark analysis with time-to-target (TTT) studies, as outlined and detailed (for example) in Ref. [43]. TTT plots have been shown to be useful in the comparison of different algorithms, and have been widely used as a tool for algorithm design and comparison [44]. As described in Ref. [44], on the ordinate axis TTT plots display the probability that an algorithm will find a solution at least as good as a given target value within a given runtime (shown on the abscissa axis), thereby characterizing expected runtimes of *stochastic* algorithms within empirical (cumulative) probability distributions. We extract these by measuring CPU runtimes needed to find a solution with an objective function value at least as good as a fixed target value for a given problem instance. To this end, we run a given algorithm n_{shots}

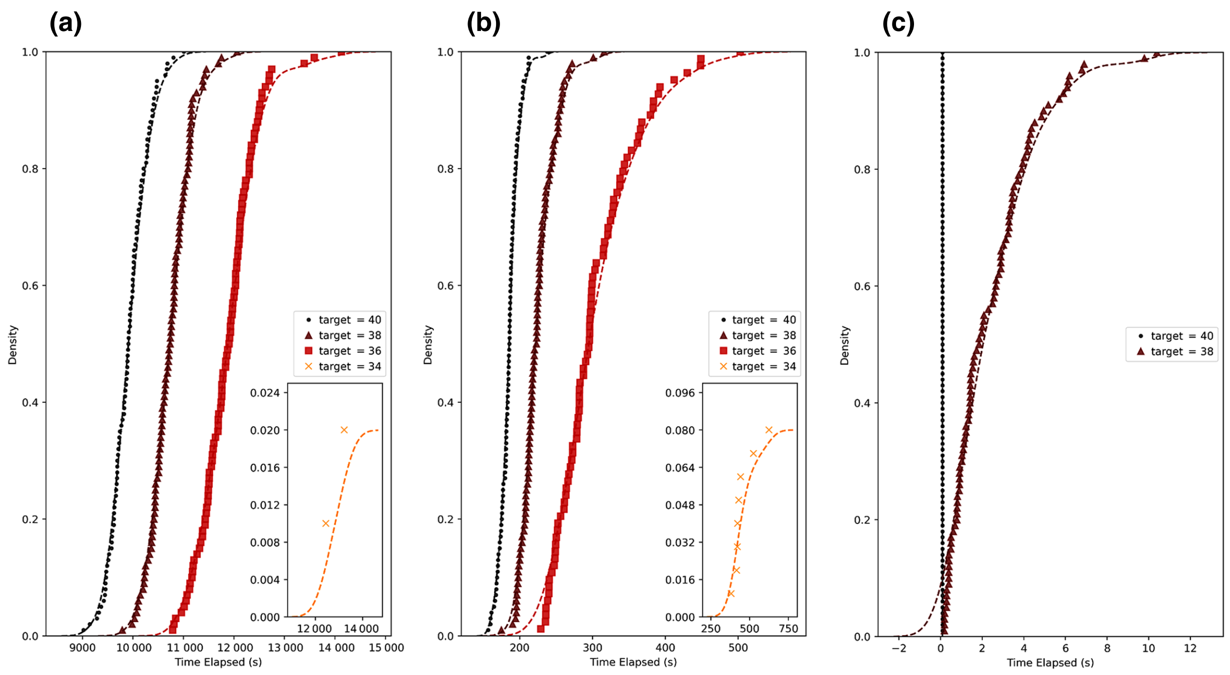

FIG. 6. Time-to-target (TTT) plots for (a) BRKGA, (b) RKO-DA, and (c) greedy solvers on the benchmark L data set with $n_{\text{seams}} = 52$. Target values (measured in seconds) have been set to 40 (poor solution quality; black circles), 38 (brown triangles), 36 (red squares), and 34 (high solution quality; orange crosses); compare Table II. Plots for a target of 34 (if available) are placed in the insets for clarity. We plot the probability to hit a fixed target cost value on the vertical axis, as a function of the algorithm's runtime on the horizontal axis. Each algorithm has been run 100 times to convergence with fixed hyperparameters, varying the seed on each run, and tracking every intermediate step. The greedy solver is unable to reach or exceed a target of 37 (the best greedy solution is 37.05), while both the BRKGA and RKO-DA solvers reach a target of 34, with probabilities about 2% for BRKGA and about 8% for RKO DA. Further details are given in the text.

times, with a distinct seed for every run (thus giving independent runs). Our results are displayed in Fig. 6, showing TTT plots for BRKGA, RKO-DA, and greedy solvers on benchmark L with $n_{\text{shots}} = 100$ and several target values. The results show that each algorithm will take longer to hit a given target as this target cost becomes smaller. Furthermore, the greedy solver is unable to reach or exceed a target of 37 (the best greedy solution is 37.05), while both the BRKGA and RKO-DA solvers reach a target of 34 with probabilities about 2% for BRKGA and about 8% for RKO DA. Finally, the expected runtime across solvers, for any given target value, is different by orders of magnitude: greedy is fastest, RKO DA is next fastest, and BRKGA is slowest. For example, for the RKO-DA solver, a runtime of approximately 300 s is sufficient to find a target solution with a cost of 36 (or better) with approximately 50% success probability, while BRKGA needs about 1.2×10^4 s to achieve the same and the greedy algorithm completely fails to achieve this solution quality.

V. CONCLUSION AND OUTLOOK

In summary, we show how to solve robot-trajectory planning problems at industry-relevant scales. To this end, we develop an end-to-end optimization pipeline that integrates classical random-key algorithms with quantum annealing into a quantum-ready, future-proof solution to the problem. With the help of a distinct separation of problem-independent and problem-dependent modules, our approach achieves an efficient problem representation that provides a native encoding of constraints, while ensuring flexibility in the choice of the underlying optimization algorithm. We provide numerical benchmark results for industry-scale data sets, showing that our approach consistently outperforms greedy baseline results. We complement this analysis with a transparent assessment of the capabilities of today's quantum hardware and resource estimates for the number of qubits required to implement a quantum native problem formulation for industry-relevant scales.

Finally, we highlight possible extensions of research going beyond our present work, where we have focused on settings involving a *single* robot, as relevant for situations in which collisions between robots can be avoided through the definition of appropriate bounding boxes. In future work it will be instructive how to generalize our random-key approach towards *multirobot* problems. To this end, we make use of an apparent analogy to the VRP as outlined in Sec. II. Specifically, we can associate individual robots with vehicles in the traditional VRP. With a straightforward extension of our formalism we can then solve both the allocation and the sequencing problems within one unified framework. For example, consider a setup with $n_{\text{seams}} = 6$ and two robots, and a sample random-key vector $\mathcal{X} = (0.25, 0.19, 0.67, 0.98, 0.04, 0.82, \mathbf{0.23}, \mathbf{0.71})$. Here, the first $n_{\text{seams}} = 6$ keys correspond to seams to be sealed by $v = 2$ robots. Along the lines of the VRP presented above, this random-key vector translates into a solution where the first robot leaves its home position and visits seams $6, 4, 5, 2$ (in this order), and then returns to its base, while the second robot leaves its home position and visits locations $1, 3$ before returning to its base. The decoder then computes the corresponding cost value by adding individual cost values, in addition to potential large penalties if the proposed solution incurs a collision between robots. Through the feedback mechanism inherent to our approach, the latter will steer the algorithm towards collision-free solutions over the course of the evolution.

ACKNOWLEDGMENTS

We would like to thank Alexander Opfolter and Cory Thigpen for management of this collaboration between AWS and BMW. The AWS team thanks Shantu Roy for his invaluable guidance and support.

APPENDIX A: EXAMPLE DECODER DESIGN

In this section, we provide the core code block (in python) for our example decoder design.

```python
def decode_piecewise(self, chromosome):
    # Split chromosome into (N) dimensions, to be decoded independently
    chr_pieces = np.array_split(chromosome, self.instance.num_dims)

    decoded_pieces = []
    sort_order = None
    for idx, piece in enumerate(chr_pieces):
        if idx == 0: # assume dim0 = abstract node number (e.g. seam number)
            # Sort in ascending order and use the order of indices
            permutation = np.argsort(piece)
            # Track seam order to pair correctly with other dimensions
            sort_order = copy(permutation)
        else:
            # Assume categorical values for all other dimensions
            n_bins = self.instance.dim_sizes[idx]
            step_size = 1. / n_bins
            # Define bin edges
            bins = np.arange(0.0, 1.0+step_size, step=step_size)
            # Assign values to bins, report bin assignment; offset by 1
            permutation = np.digitize(piece, bins) - 1
            # Rearrange output according to seam argsort order
            permutation = permutation[sort_order]

        decoded_pieces.append(permutation)

    # Pair elements in same position across dimensions
    # i.e. [1,2,3], [10,20,30] -> [1,10], [2,20], [3,30]
    decoded = list(zip(*decoded_pieces))
    return decoded

def decode(self, chromosome, rewrite: bool = False) -> float:
    # Decode chromosome into tour, with nodes of N-dimensions
    decoded = self.decode_piecewise(chromosome)

    # Add distance from LAST node to FIRST node
    cost = self.instance.distance(decoded[-1], decoded[0])

    # Cumulative sum of distances for intermediate nodes of tour
    for i in range(len(decoded)-1):
        cost += self.instance.distance(decoded[i], decoded[i + 1])

    return cost
```

Listing 1. Core code block of example decoder.

APPENDIX B: HYPERPARAMETERS FOR NUMERICAL EXPERIMENTS

In this section, we provide details for the specific model configurations (hyperparameters) as used to solve benchmark instances L and XL with the BRKGA and RKO-DA solvers, respectively.

TABLE III. Optimal hyperparameter values (rounded to four decimal points) for the BRKGA solver on benchmarks L and XL.

Benchmark	elite_ percentage	mutants_ percentage	num_ generations	patience	population_ size	seed	num_elite_ parents	total_ parents
L	0.4465	0.0518	2000	52	9918	839	2	3
XL	0.4894	0.2594	2000	66	7978	263	2	3

TABLE IV. Optimal hyperparameter values (rounded to four decimal points) for the RKO-DA solver on benchmarks L and XL.

Benchmark	maxiter	seed	visit	accept	initial_temp	restart_temp_ratio
L	5547	151	1.1321	−2.3875	20 314.2789	6.3192×10^{-5}
XL	27 635	656	1.1741	−0.4968	49 061.6875	1.1119×10^{-4}

[1] M. Muradi and R. Wanka in *2020 6th International Conference on Control, Automation and Robotics (ICCAR)* (IEEE, Singapore, 2020), p. 130.

[2] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, and P. Bunyk, *et al.*, Quantum annealing with manufactured spins, Nature **473**, 194 (2011).

[3] P. Bunyk, E. Hoskinson, M. W. Johnson, E. Tolkacheva, F. Altomare, A. J. Berkley, R. Harris, J. P. Hilton, T. Lanting, and J. Whittaker, Architectural considerations in the design of a superconducting quantum annealing processor, IEEE Trans. Appl. Supercond. **24**, 1 (2014).

[4] H. G. Katzgraber, Viewing vanilla quantum annealing through spin glasses, Quantum Sci. Technol. **3**, 030505 (2018).

[5] P. Hauke, H. G. Katzgraber, W. Lechner, H. Nishimori, and W. Oliver, Perspectives of quantum annealing: Methods and implementations, Rep. Prog. Phys. **83**, 054401 (2020).

[6] J. R. Finžgar, P. Ross, J. Klepsch, and A. Luckow, QUARK: A framework for quantum computing application benchmarking, ArXiv:2202.03028 (2022).

[7] A. Luckow, J. Klepsch, and J. Pichlmeier, Quantum computing: Towards industry reference problems, Digitale Welt **5**, 38 (2021).

[8] J. Klepsch, J. Kopp, A. Luckow, H. Weiss, B. Standen, D. Vozl, C. Utschig-Utschig, M. Streif, T. Strohm, and H. Ehm, *et al.*, Industry quantum applications, https://www.qutac.de/wp-content/uploads/2021/07/QUTAC`Paper.pdf (2021).

[9] J. F. Gonçalves and M. G. C. Resende, Biased random-key genetic algorithms for combinatorial optimization, J. Heurist. **17**, 487 (2011).

[10] J. C. Bean, Genetic algorithms and random keys for sequencing and optimization, ORSA J. Comput. **6**, 154 (1994).

[11] R. M. A. Silva, M. G. C. Resende, and P. M. Pardalos, Finding multiple roots of a box-constrained system of nonlinear equations with a biased random-key genetic algorithm, J. Global Optim. **60**, 289 (2014).

[12] W. M. Spears and K. A. DeJong, in *Proceedings of the Fourth International Conference on Genetic Algorithms* (Morgan Kaufmann Publishers, Burlington, Massachusetts, 1991), p. 230.

[13] R. F. Toso and M. G. C. Resende, A C++ application programming interface for biased random-key genetic algorithms, Optim. Methods Softw. **30**, 81 (2015).

[14] C. E. Andrade, R. F. Toso, J. F. Gonçalves, and M. G. C. Resende, The multi-parent biased random-key genetic algorithm with implicit path-relinking and its real-world applications, Eur. J. Oper. Res. **289**, 17 (2021).

[15] M. G. C. Resende, R. F. Toso, J. F. Gonçalves, and R. M. A. Silva, A biased random-key genetic algorithm for the Steiner triple covering problem, Optim. Lett. **6**, 605 (2012).

[16] J. F. Gonçalves and M. G. Resende, A parallel multi-population biased random-key genetic algorithm for a container loading problem, Comput. Oper. Res. **39**, 179 (2012).

[17] M. Luby, A. Sinclair, and D. Zuckerman, Optimal speedup of Las Vegas algorithms, Inf. Process. Lett. **47**, 173 (1993).

[18] M. G. C. Resende, L. Morán-Mirabal, J. L. González-Velarde, and R. F. Toso, Restart strategy for biased random-key genetic algorithms, http://mauricio.resende.info/doc/brkga-restart.pdf (2013).

[19] M. L. Lucena, C. E. Andrade, M. G. C. Resende, and F. K. Miyazawa, in *Proceedings of the XLVI Symposium of the Brazilian Operational Research Society* (Sociedade Brasileira de Pesquisa Operacional (SOBRAPO), Rio de Janeiro, Brazil, 2014).

[20] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, and J. Bright, *et al.*, SciPy 1.0: Fundamental algorithms for scientific computing in Python, Nat. Methods. **17**, 261 (2020).

[21] C. Tsallis and D. A. Stariolo, Generalized simulated annealing, Phys. A: Stat. Mech. Appl. **233**, 395 (1996).

[22] Y. Xiang, D. Sun, W. Fan, and X. Gong, Generalized simulated annealing algorithm and its application to the Thomson model, Phys. Lett. A **233**, 216 (1997).

[23] Y. Xiang, S. Gubian, B. Suomela, and J. Hoeng, Generalized simulated annealing for global optimization: The GenSA package, R Journal **5**, 13 (2013).

[24] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, A limited memory algorithm for bound constrained optimization, SIAM J. Sci. Comput. **16**, 1190 (1995).

[25] A. Lucas, Ising formulations of many NP problems, Front. Phys. **2**, 5 (2014).

[26] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lu, H. Wang, and Y. Wang, The unconstrained binary quadratic programming problem: A survey, J. Comb. Optim. **28**, 58 (2014).

[27] F. Glover, G. Kochenberger, and Y. Du, Quantum bridge analytics I: A tutorial on formulating and using QUBO models, 4OR **17**, 335 (2019).

[28] M. Anthony, E. Boros, Y. Crama, and A. Gruber, Quadratic reformulations of nonlinear binary optimization problems, Math. Program. **162**, 115 (2017).

[29] E. Ising, Beitrag zur theorie des ferromagnetismus, Z. Phys. **31**, 253 (1925).

[30] T. Kadowaki and H. Nishimori, Quantum annealing in the transverse Ising model, Phys. Rev. E **58**, 5355 (1998).

[31] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda, A quantum adiabatic evolution algorithm applied to random instances of an NP-complete problem, Science **292**, 472 (2001).

[32] S. V. Isakov, G. Mazzola, V. N. Smelyanskiy, Z. Jiang, S. Boixo, H. Neven, and M. Troyer, Understanding Quantum Tunneling through Quantum Monte Carlo Simulations, Phys. Rev. Lett. **117**, 180402 (2016).

[33] V. Choi, Minor-embedding in adiabatic quantum computation. I: The parameter setting problem, Quantum Inf. Process. **7**, 193 (2008).

[34] A. Zaribafiyan, D. J. J. Marchand, and S. S. Changiz Rezaei, Systematic and deterministic graph minor embedding for Cartesian products of graphs, Quantum Inf. Process. **16**, 1 (2017).

[35] A. Perdomo-Ortiz, A. Feldman, A. Ozaeta, S. V. Isakov, Z. Zhu, B. O'Gorman, H. G. Katzgraber, A. Diedrich, H. Neven, and J. de Kleer, *et al.*, On the readiness of quantum optimization machines for industrial applications, arXiv:1708.09780 (2017).

[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi Jr., Optimization by simulated annealing, Science **220**, 671 (1983).

[37] F. Glover, in *Artificial evolution*, Lecture Notes in Computer Science, vol. 1363 (Springer, Berlin, 1997), p. 13.

[38] M. G. C. Resende, C. C. Ribeiro, F. Glover, and R. Marti, in *Handbook of Metaheuristics*, International Series in Operations Research and Management Science, vol. 146 (Springer, New York, NY, 2010), p. 87.

[39] A. W. S. Braket, Amazon braket python SDK: A python SDK for interacting with quantum devices on Amazon braket, https://github.com/aws/amazon-braket-sdk-python (2021).

[40] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, ROS: An open-source robot operating system, https://ros.org (last accessed: May 2022).

[41] S. Karaman and E. Frazzoli, Sampling-based algorithms for optimal motion planning, Int. J. Rob. Res. **30**, 846 (2011).

[42] J. Bergstra, D. Yamins, and D. Cox, in *Proceedings of the 30th International Conference on Machine Learning*, Proceedings of Machine Learning Research, vol. 28, edited by S. Dasgupta and D. McAllester (PMLR, Atlanta, Georgia, USA, 2013), p. 115–123, https://proceedings.mlr.press/v28/bergstra13.html.

[43] M. G. C. Resende and C. C. Ribeiro, in *Greedy Randomized Adaptive Search Procedures*, Handbook of metaheuristics (Kluwer Academic Publishers, Boston, 2003), p. 219.

[44] R. M. Aiex, M. Resende, and C. Ribeiro, TTT plots: A perl program to create time-to-target plots, Optim. Lett. **1**, 355 (2007).