IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 22 February 2024, accepted 24 March 2024, date of publication 29 March 2024, date of current version 4 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3383313

RESEARCH ARTICLE

Parallel Quantum Rapidly-Exploring Random Trees

PAUL LATHROP[1,2], (Student Member, IEEE), BETH BOARDMAN[2], AND SONIA MARTÍNEZ[1], (Fellow, IEEE)

[1]Department of Mechanical and Aerospace Engineering, University of California at San Diego, San Diego, CA 92092, USA
[2]Los Alamos National Laboratory, Los Alamos, NM 87545, USA

Corresponding author: Paul Lathrop (pdlathrop@gmail.com)

This work was supported by Los Alamos National Laboratory and is approved for release under LA-UR-23-31988v3.

ABSTRACT In this paper, we present the Parallel Quantum Rapidly-Exploring Random Tree (Pq-RRT) algorithm, a parallel version of the Quantum Rapidly-Exploring Random Trees (q-RRT) algorithm. Parallel Quantum RRT is a parallel quantum algorithm formulation of a sampling-based motion planner that uses Quantum Amplitude Amplification to search databases of reachable states for addition to a tree. In this work we investigate how parallel quantum devices can more efficiently search a database, as the quantum measurement process involves the collapse of the superposition to a base state, erasing probability information and therefore the ability to efficiently find multiple solutions. Pq-RRT uses a manager/parallel-quantum-workers formulation, inspired by traditional parallel motion planning, to perform simultaneous quantum searches of a feasible state database. We present symbolic runtime comparisons between parallel architectures, then results regarding likelihoods of multiple parallel units finding any and all solutions contained with a shared database, with and without reachability errors, allowing efficiency predictions to be made. We offer simulations in dense obstacle environments showing efficiency, density/heatmap, and speed comparisons for Pq-RRT against q-RRT, classical RRT, and classical parallel RRT. We then present Quantum Database Annealing, a database construction strategy that uses a temperature construct to define database creation over time for balancing exploration and exploitation.

INDEX TERMS Sampling-based motion planning, quantum computing, parallel computing.

I. INTRODUCTION

Quantum computing algorithms have shown promise in accelerating the search for solutions when applied to computationally intensive, complex problems. More efficient solutions have been found and proven with quantum computing in fields such as pure science simulations [2], cryptography [3], and machine learning [4].

With respect to robotics and motion planning, quantum algorithms have also been found to increase speed and efficiency. The heart of quantum advantage is derived from quantum parallelism and the ability to perform simultaneous computations on superpositions of states, which motivated our work in [1]. To aid in sampling-based motion planning, the key efficiency bottleneck is the search for dynamically-reachable, obstacle-free states to connect to the search tree. Unstructured databases of possible next states can be searched simultaneously with Quantum Amplitude Amplification (QAA) to efficiently find an amenable state, but the quantum measurement process forces information loss through the collapse of the superposition. Although all database states are searched in parallel, the process can only return one state. In this work we are motivated by efforts in traditional motion planning to parallelize sampling-based planners for efficient path generation using multi-core computers and GPUs, and the ability of quantum algorithms to perform parallel computations. We seek to explore ways for quantum motion planning algorithms to allow multiple, parallel quantum computers to more efficiently search a database of states and return multiple possible solutions.

The associate editor coordinating the review of this manuscript and approving it for publication was Tao Liu.

A. LITERATURE REVIEW

In this section, we provide an overview of quantum computing as it applies to robotic applications, non-quantum efforts to parallelize sampling-based motion planning algorithms, the use of annealing and temperature constructs as they applies to motion planning, and how this is related to our efforts to increase the efficiency of q-RRT.

Quantum algorithms have been applied to a range of robotic and motion planning-related applications. They have been used to solve generalized optimization problems [5], estimate stochastic processes [6], and provide speedup to Monte Carlo methods [7]. They have also performed quantum searches [8] of physical regions [9], found marked elements of a Markov chain [10], and searched more abstract spaces such as a search engine network [11]. A more detailed overview on how quantum computing has been applied to robotics, along with open questions, can be found at [12].

Quantum computing has also been used to improve motion planning specifically. Quantum reinforcement learning [13] has increased the speed and robustness (when compared to classical, non-quantum algorithms) of robotic reinforcement learning algorithms when learning optimal policies in gridded environments [14], [15]. An additional use of quantum computing for robotic trajectory planning is addressed in [16], which uses a Quantum Evolutionary Algorithm to search for optimal trajectories with a population dynamics/mutation quantum algorithm. Lastly, the review [17] examines quantum control algorithms, and the topic of feedback control accomplished using quantum computing. The work at hand is distinct from the state of the art of quantum computing as applied to motion planners, as we apply such methods to sampling-based planners, which have the advantage of providing fast solutions in high dimensional environments alongside providing probabilistic completeness guarantees [18]. Besides our previous q-RRT algorithm, quantum computing has not, to the best of our knowledge, been applied to sampling-based planning algorithms. A further overview on how quantum computing has been applied to motion planning and robotics can be found in [1].

In the field of non-quantum motion planning, sampling-based planning algorithms such as Probabilistic Roadmaps (PRMs) [19] and Rapidly-exploring Random Trees (RRT) [20] have taken the forefront due to their efficiencies in high-dimensional planning spaces and ability to handle complex robot dynamic constraints [21], such as robotic grasping tasks, autonomous vehicle planning, and UAV navigation. RRTs and PRMs have been extended in many ways to improve their sampling speed, exploration ability, and collision-checking subroutine. RRT* is an important extension regarding path optimality through rewiring [22]. An overview of sampling-based motion planning can be found in the textbook [18]. Three ways to increase motion planning efficiency are the use of quantum computing, the use of parallel algorithms and architectures, and the use of sampling strategies. In this work we consider the combinations of the three approaches through parallel quantum computing and database construction strategies, which is akin to sampling strategies in classical algorithms.

Motion planning algorithms have been written for parallel tree creation [23] and parallel computation with GPUs [24]. In [23], local trees are built in parallel to explore difficult regions, and guidelines on when to create and grow local trees are made. In [24], the authors parallelize the collision-checking procedure using GPUs to increase optimal planning speed in high-dimensional spaces. The work [25] outlines Parallel RRT and Parallel RRT*, which uses lock free parallelism and partition based sampling to provide superlinear speedup to RRT and RRT*. The work [26] compares parallel versions of RRTs on large scale distributed memory architectures, including *or*-Parallel RRT (multiple simultaneous individual RRT's) and two methods of collaborative single RRT, Distributed RRT and Manager-Worker RRT. The work [26] also includes a succinct literature review regarding parallel motion planning and Parallel RRT. For comparison purposes, in the work at hand we consider a class of Manager-Worker Parallel RRTs, focusing on the parallelization of the collision-checking procedure of RRT. This is justified by the findings in [26], which concludes that for variable expansion cost cases, where the communication load is insignificant compared to the computation load, Manager-Worker RRT outperforms, or nearly matches, Distributed RRT in studied passage, corridor, and roundabout environments.

The work [27] discusses parallel quantum computing architectures and control strategies for distributed quantum machines, noting that multiple few-qubit devices may be more technologically feasible than single many-qubit devices. In this work, we consider parallel quantum computers to be multiple simultaneous identical quantum devices governed by a classical device in order to perform parallel computation.

A second extension to increase the efficiency of q-RRT relies on database construction, where we employ a method inspired by simulated annealing. Simulated annealing [28] is an optimization technique that relies on decreasing a temperature parameter to find global maxima and minima of a nonconvex optimization problem, which is somewhat robust to local features. Temperature acts as a guide to the probability of accepting a worse state, allowing an optimizer to explore past local features while eventually settling on estimates of global optima when temperature falls. An overview can be found at [29] and [30].

Although a temperature construct is mainly used in the context of optimization, similar annealing ideas have also been applied to motion planning, and we intend to apply them to guiding the exploration vs exploitation trade-off of the planning algorithm. In a manner somewhat reminiscent of annealing, the work [31] uses a dynamic reaction-diffusion process to propagate, then contract, a search area for

a goal location. Additionally, the covariant Hamiltonian optimization for motion planning (CHOMP) method [32] uses gradient techniques to improve trajectories and solve motion planning queries. CHOMP uses simulated annealing to avoid local minima in trajectory optimization and not to guide sampling-based motion planning itself. The work [33] uses simulated annealing to balance exploitation of Sampling-Based A* (SBA*) and exploration of Rapidly-exploring Random Tree* (RRT*). As cooling occurs, the probability of choosing the exploration strategy drops and the probability of choosing the exploitation strategy increases. Similarly, the transition based RRT [34], [35] method uses a temperature quantity inspired by simulated annealing to define the difficulty level of transition tests to accept higher cost configurations in an effort to explore a configuration space. Similar to this work, we use a temperature quantity to guide the level of exploration, but because we are using quantum computing with q-RRT to perform motion planning, temperature factors into database construction rather than individual samples themselves.

In [1], we introduced how quantum computing methods can be applied to sampling-based motion planning in two ways, a full path database search and an RRT-based single-state database search q-RRT. The Quantum Rapidly-Exploring Random Trees algorithm, q-RRT, uses Quantum Amplitude Amplification (QAA) to search databases of possible reachable states. A focus of our work [1] was in estimating solution likelihood (so QAA could be performed an optimal number of times) through the use of random square lattice environments and numerical simulations. We chose this approach over quantum counting in an effort to keep oracle efficiency high. In the work at hand, we shift focus to address a particular shortcoming of using quantum computers and qubits: these approaches suffer from the limits of quantum mechanics. When qubit measurement decoheres a superposition into a definite state, information is lost. Additionally, qubits cannot be copied, as this is akin to quantum measurement. We study how quantum devices, working in parallel, can efficiently solve motion planning problems, while generalizing environments away from random square lattices. Instead of focusing on how many solutions exist within a database (which can be found with the Quantum Counting Algorithm (QCA)), we focus on how multiple solutions can efficiently be found from a single database. This work is based on Chapter 6 of the first author's Ph.D. thesis [36], with additional analysis and editing.

B. CONTRIBUTIONS

The main contributions of this work are the following.

- Creation of a parallel quantum computing variation to q-RRT, called Parallel q-RRT (Pq-RRT), which uses a parallel quantum computing structure to allow multiple solutions from a single database in general obstacle environments;
- Symbolic runtime analysis of shared and unshared database formulations of a Manager-Worker parallel architecture;
- Characterization of key probability values for multiple quantum workers searching a shared database, with and without false positive and false negative oracle errors in order to minimize efficiency loss;
- Creation of a construction strategy for quantum-search databases, called Quantum Database Annealing, which uses a temperature construct to select sample distances and balance exploration vs exploitation;
- Demonstration (through simulation) of the increased efficiency of Pq-RRT over q-RRT, as compared to the efficiency increase of Parallel RRT over RRT;
- Simulations of faster tree exploration with Quantum Database Annealing as compared to standard uniform-sampling database construction.

II. ORGANIZATION AND NOTATION

In Section III-A, for reference purposes we provide a working definition of q-RRT from [1]. In Sections III-B and III-D, we define Parallel q-RRT then key probability results for Pq-RRT, respectively. In Section III-E, we define the database construction strategy Quantum Database Annealing. In Section IV-A, we provide runtime and efficiency results for q-RRT and Pq-RRT as compared to RRT and Parallel RRT. In section IV-B, we provide heatmaps of q-RRT's node placement speed over RRT, and in Section IV-C we provide narrow corridor results for q-RRT. In Section IV-D, we provide tree comparisons between Quantum Database Annealing and standard database construction.

A. NOTATION

The set of natural numbers is denoted as $\mathbb{N}$, and similarly, $\mathbb{R}$ is the set of real numbers, while $\mathbb{C}$ is the set of complex numbers. For $d, p \in \mathbb{N}$, we use $\mathbb{R}^d$ to refer to d-dimensional real vector space, and by $x \in \mathbb{R}^d$ a vector in it. Lastly, the Euclidean norm in $\mathbb{R}^d$ is identified with $\|\cdot\|_2$.

B. QUANTUM COMPUTING BASICS

We briefly introduce quantum computing basics and how we use quantum algorithms to solve motion planning problems. We defer a more in depth introduction of quantum computing basics to [1] and a comprehensive circuit-level discussion to [37]. Let $|z\rangle$ refer to the quantum state represented by the qubit z. Quantum computers encode information in basic units called qubits, given as the superposition of two basis quantum states, $|0\rangle$ and $|1\rangle$. The set $\{|0\rangle, |1\rangle\}$ defines a basis of quantum states. Qubits maintain probability amplitudes (relative measurement likelihoods) α and β. A qubit $|\Psi\rangle$ can exist in a superposition of $|0\rangle$ and $|1\rangle$, of the form $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ with $\alpha, \beta \in \mathbb{C}$, $|\alpha|^2 + |\beta|^2 = 1$. Multiple qubits come together to form a memory storage unit called a qubit register. The measurement process involves the collapse of the state $|\Psi\rangle$ (described by a superposition) to a base state

$\{|0\rangle, |1\rangle\}$ (described by a definite classical state) according to probabilities α^2 and β^2 (known as the Born rule).

Quantum algorithms perform fast parallel computations on superpositions in a process known as quantum parallelism [15]. This manipulates the amplitudes α and β of the system, which cannot be known explicitly, as any measurement collapses the qubit. Previously we leveraged quantum parallelism to create a new algorithm for solving motion planning problems.

Our Quantum Rapidly-Exploring Random Tree (q-RRT) algorithm [1] uses Quantum Amplitude Amplification (QAA) with a Boolean oracle function $\mathcal{X}$ that evaluates reachability of states. QAA uses an oracle $\mathcal{X}$ to increase the probability of measuring a selected ('good') state Ψ (such that $\mathcal{X}(\Psi) = 1$). The QAA precise definition, mechanism of action, and discussion can be found at [38], page 56. Similar to [1], we take advantage of QAA to quantum search a size-N unordered database for oracle-tagged items in $\mathcal{O}(N^{1/2})$ oracle calls, whereas non-quantum search algorithms require $\mathcal{O}(N)$ calls.

III. PARALLEL QUANTUM RRT AND QUANTUM DATABASE ANNEALING

In Section III-A, we outline the q-RRT Algorithm from [1], followed by a presentation of the Parallel Quantum RRT Algorithm in Section III-B, and then probability results for Pq-RRT in Section III-D. Lastly, in Section III-E, we present the Quantum Database Annealing strategy.

A. QUANTUM RRT ALGORITHM

The q-RRT algorithm from our previous work [1] is a tree-based search algorithm based on RRTs [39]. Quantum RRT uses QAA on a database of possible parent-child pairs to admit reachable points to the tree. In this work, q-RRT returns a path (as opposed to the full tree in [1]) and has the end condition of finding a goal. The line-by-line algorithm can be found at [1], and because of q-RRT's similarities with Pq-RRT Manager (Alg. 1) and Pq-RRT Worker (Alg. 2), we omit the line-by-line and instead reference lines of the latter two enumerated algorithm descriptions.

The q-RRT Algorithm takes as input an initial state x_0 and a goal state x_G in a compact configuration space $C \subseteq \mathbb{R}^d$, a number of qubit registers n, and a quantum oracle function $\mathcal{X}$. It returns a dynamically feasible obstacle free path γ. The q-RRT algorithm adds nodes to graph T until there is a node within distance δ of the goal x_G. To add a node, q-RRT creates a 2^n sized database D of random possible nodes and the nearest parent in T to the random node, as shown in Alg. 1 on lines 4-8. A $1-$to-1 mapping F is created between database D and qubit $|\Psi\rangle$ (shown in Alg. 2, line 1). Then, the qubit is initialized and an equal superposition between states set (shown in Alg. 2 on lines 2 and 3 respectively). Let $\mathbf{W}$ be the Walsh-Hadamard transform, the operator which maps a qubit to an equal superposition of all qubit states. On lines 4-6 of Alg. 2, the operator Q performs QAA to amplify the probability amplitudes of correct states as defined by oracle $\mathcal{X}$, which tests the reachability of random samples to the nearest proposed parent P of the existing graph T, thus ensuring that T is fully reachable. For an analysis into selecting $i_{\max}$ we refer readers to [1].

Measurement is performed and the correct database element selected in Alg. 2 on line 7. After measurement is performed, the quantum state has collapsed into a definite state and no further information (beyond the definite state) can be gained from the qubit. Lines 14-16 of Alg. 1 allow a node placed within δ of x_G to be admitted to T as x_G, ending the algorithm. The path is returned after successful loop execution on Alg. 1, line 19.

B. PARALLEL QUANTUM RRT

In this section, we define the Parallel Quantum RRT (Pq-RRT) algorithm as a manager (Alg. 1) worker (Alg. 2) formulation. The Pq-RRT algorithm performs reachability tests using a parallel pool of quantum computers, and is a direct extension of q-RRT inspired by parallel motion planning. The manager algorithm assigns work to the parallel pool and adds results to the tree T. The assigned work consists of each quantum worker performing a reachability check on a database D using QAA with a quantum oracle, and returning a single database element. The specific parallelization architecture is chosen for a few reasons. We consider scenarios where generally worker runtime cost dominates the message passing cost (as per [26]). This rules out such architectures as disjoint workers independently searching for a solution by growing separate trees, which have relatively little message passing but are much less runtime-efficient in finding a solution.

In the chosen manager-worker scheme, instead of discretizing a search space to allow workers to each grow a separate part of a tree, each worker is tasked with adding a single element to the tree (anywhere). This removes the idleness aspect of workers, as workers do not have to be actively listening for tree updates and do not rely on the work of others to perform their own search. Additionally, because of the probabilistic nature of the quantum search process, the parallel quantum routine generally can have all workers complete work simultaneously. This feature is not possible in non-quantum parallel architectures, as each worker is performing a stochastic search for a solution, which generally takes differing times between workers. In the quantum architecture, however, the runtime to amplify a database is much more consistent, and if each quantum worker is performing the same number of amplifications before measurement, they should complete a search nearly simultaneously. The key difference lies in the goal of the work performed. A solution does not need to be deterministically found for work to be completed (as in the non-quantum case). Work is instead completed when a solution is more likely to be found, which can be standardized for runtime across the workers.

Alg. 1, the manager algorithm, has the same inputs and outputs as q-RRT. The worker algorithm, Alg. 2, admits as

inputs the current tree T, the number of qubit registers n, the quantum oracle function $\mathcal{X}$, and a copy of the shared database D, and returns a selected element of the database $[x_{\text{add}}, P]$.

Algorithm 1 Pq-RRT Manager, Shared Database

Input: $x_0, x_G\ n,$ oracle $\mathcal{X}$
Output: Path γ

1: Init p-worker pool
2: Init tree T with root at x_0
3: **while** $x_G \notin T$ **do**
4: **for** $i = 1$ to 2^n **do**
5: $t =$ random point
6: $P =$ closest parent of t in T
7: $D(i) = [t;\ P]$
8: **end for**
9: $[x_{\text{add}}, P](k) =$ Worker$(T, n, \mathcal{X}, D)$, for $k \in p$
10: **for** $k = 1$ to p **do**
11: **if** $[x_{\text{add}}, P](k) \notin T$ **then**
12: Add $[x_{\text{add}}, P](k)$ to T
13: **end if**
14: **if** $\|x_{\text{add}}(k) - x_G\| < \delta$ **then**
15: $x_{\text{add}}(k) = x_G$
16: **end if**
17: **end for**
18: **end while**
19: Return path γ from T

Algorithm 2 Pq-RRT Worker, Shared Database

Input: $T, n, \mathcal{X}, D$
Output: $[x_{\text{add}}, P]$

1: Enumerate D via $F : \{0, 1\}^n \rightarrow D$
2: Init n qubit register $|z\rangle \leftarrow |0\rangle^{\otimes n}$
3: $|\Psi\rangle \leftarrow \mathbf{W}\,|z\rangle$
4: **for** $i = 1$ to $i_{\max}$ **do**
5: $|\Psi\rangle \leftarrow Q(\mathcal{X})\,|\Psi\rangle$
6: **end for**
7: $[x_{\text{add}}, P] \leftarrow F(\text{measure}(|\Psi\rangle))$
8: Return $[x_{\text{add}}, P]$

Two versions of this parallel formulation are possible, shared and unshared database. The fundamental difference is whether parallel pool workers create a database or perform QAA on copies of the same database D, which is created by the manager. For the shared database version, as shown in Fig. 1, in Alg. 1 Lines 4-8 the manager creates the database D and passes copies to each worker $k \in p$, as shown by the inputs to Alg. 2. In this way, the workers would "share" and search (copies of) the same database. The manager ignores additional identical solutions returned by different workers, which is a fast process given that the workers essentially are returning an index to a database element.

For an unshared database, as shown in Fig. 2, the database construction step is performed within the worker algorithm

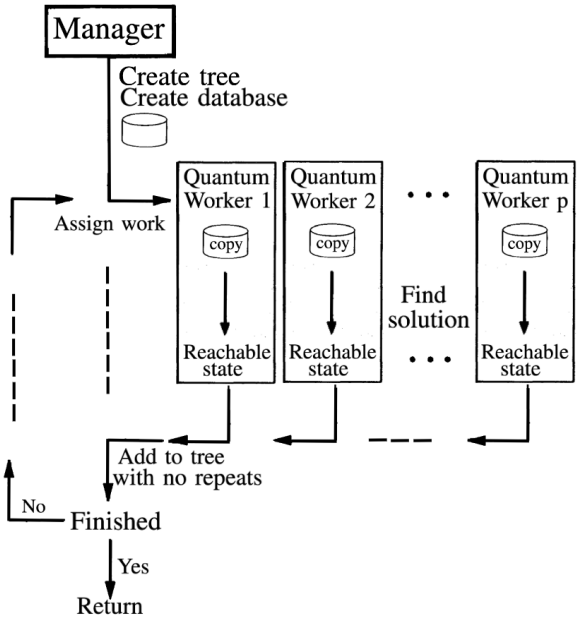

FIGURE 1. A graphical depiction of the shared database quantum parallel RRT algorithm. The manager (Alg. 1) creates a database and passes copies to the quantum workers (Alg 2).

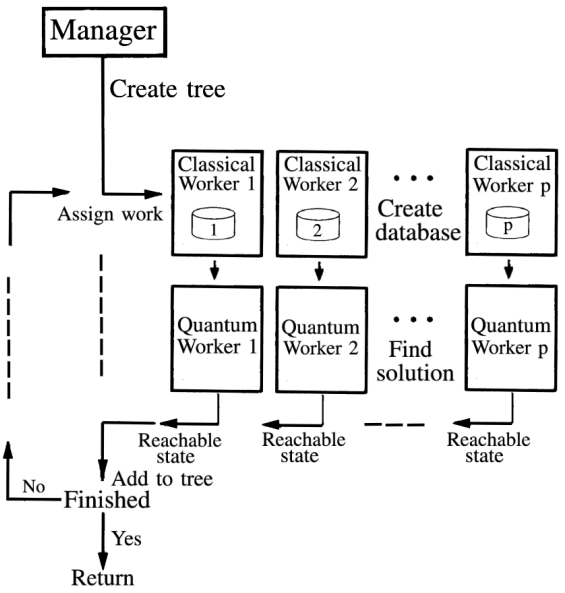

FIGURE 2. A graphical depiction of the unshared database quantum parallel RRT algorithm. The manager prompts p classical workers to create p different databases, which are passed to p quantum workers to find solutions, which are returned to the manager.

(Alg. 2), which can be a classical worker until QAA is performed.

The advantage of the shared database approach is an increase in database-use-efficiency due to extracting multiple possible reachable states per database construction. This aligns with the main motivation behind this work, which is to mitigate the probability information loss due to quantum measurement. Because quantum computers are reducing the

time spent on the computationally intensive portion of the algorithm (state collision/reachability checks), steps such as database construction will become a larger proportion of algorithm runtime, so it is advantageous to have high database-use-efficiency, which we analyze next. However, the shared database approach can be less oracle-call efficient (compared to unshared database), with fewer reachable states are admitted per oracle call because repeated identical solutions are discarded. This is shown in Section IV-A Fig. 9, and we discuss how important that efficiency loss is and how to mitigate it in Section III-D.

C. RUNTIME ANALYSIS

We consider runtime and computational cost comparisons of the shared and unshared database formulations to characterize database-use efficiency and discuss potential trade-offs.

Since databases are made from parent–child connections, adding an element to a database requires a tree search. Let the cost of randomly generating a possible node be τ_0 (generally small), and let the unit search cost of finding a parent be τ_s (generally computing some distance metric). Then, when the tree has M nodes, building a database of size 2^n (where n is the number of qubits in use) has cost $C_D(M)$,

$$C_D(M) = (\tau_0 + \tau_s M)2^n.$$

Remark 1: We note that this search cost of $\tau_s M$ can be minimized through a spatial tree discretization scheme where only partial *local* tree searches are necessary, but generally this cost still scales as a function of M.

Let the message passing costs of passing a node between manager and worker be τ_v. Then, the cost of passing an M-node tree is $M\tau_v$ and the cost of passing a 2^n-sized database is $2^{n+1}\tau_v$.

Remark 2: The $+1$ appears in the exponent of the database-passing cost due to database elements being a parent–child pair, but this can be reduced to be an *index* rather than node.

Let the cost of one application of QAA be τ_Q. Since optimal amplification is achieved by $\frac{\pi}{4}\sqrt{2^n/m}$ applications of QAA, we define the optimal amplification cost as C_Q,

$$C_Q = \left\lfloor \frac{\pi}{4}\sqrt{\frac{2^n}{m}} \right\rfloor \tau_Q,$$

where m is the number of solutions within the database.

Remark 3: In general, for traditional (non-quantum) sampling-based planning algorithms, costs like τ_Q are much larger than $\tau_{0,s,v}$. Connection costs far outweigh sampling and tree-traversal costs. Additionally, the function is floored, as QAA can only be applied a discrete number of times.

In the following, we derive two types of costs: runtime and total cost. Runtime cost, for p-core parallel algorithms, is the analog to computation time, where centralized components are counted fully and decentralized components (that run in parallel) are discounted by a factor of p. Total cost has two analogous scenarios: total power consumption and sequential algorithm runtime (to accomplish the same task), where decentralized components lose the p-fold discount factor. Terms in Props. 1 and 2 are presented in the order with which they are incurred when the algorithm is running.

Proposition 1: The runtime cost of the unshared database algorithm shown in Fig. 2, with p workers, is,

$$C^{\text{Run}}_{\text{Unsh}} = p\sum_{k=1::p}^{M} k\tau_v + \frac{M}{p}2^n\tau_0 + 2^n\sum_{k=1::p}^{M} k\tau_s + \frac{M}{p}C_Q + M\tau_v, \tag{1}$$

and the total cost is,

$$C^{\text{Tot}}_{\text{Unsh}} = p\sum_{k=1::p}^{M} k\tau_v + M2^n\tau_0 + 2^n p\sum_{k=1::p}^{M} k\tau_s + MC_Q + M\tau_v, \tag{2}$$

where the notation $\sum_{k=1::p}^{M}$ refers to the sum from $k = 1$, stepping by p, to M, while $k < M$.

For the runtime cost in Eq. (1), the first term, $p\sum_{k=1::p}^{M} k\tau_v$, is the message passing cost of the current tree, of growing size, being sent to the p workers. Since the tree is growing by p nodes each iteration, the message passing cost is made of a sum and must be incurred p times, as the tree must be sent to each worker each iteration. The second term, $\frac{M}{p}2^n\tau_0$, is the random sampling cost incurred in database construction by each worker, and since this process is distributed, at runtime the M multiplier is scaled by $\frac{1}{p}$. The third term, $2^n\sum_{k=1::p}^{M} k\tau_s$, is the parent search cost, which grows as the tree grows by p nodes each iteration. The fourth term, $\frac{M}{p}C_Q$, is the cost of applying QAA (and testing for reachability), and is distributed at runtime (divided over p). The last term, $M\tau_v$, is the cost of sending all M found nodes back to the manager.

To attain the total cost in Eq. (2), the distributed terms of Eq. (1) (terms 2, 3, and 4) are multiplied by p to find sequential runtime cost.

Proposition 2: The runtime cost of the shared database algorithm shown in Fig. 1, with p workers, is,

$$C^{\text{Run}}_{\text{Sh}} = \frac{M}{p}2^n\tau_0 + 2^n\sum_{k=1::p}^{M} k\tau_s + M2^{n+1}\tau_v + \frac{M}{p}C_Q + M\tau_v, \tag{3}$$

and the total cost is,

$$C^{\text{Tot}}_{\text{Sh}} = \frac{M}{p}2^n\tau_0 + 2^n\sum_{k=1::p}^{M} k\tau_s + M2^{n+1}\tau_v + MC_Q + M\tau_v, \tag{4}$$

where the notation $\sum_{k=1::p}^{M}$ refers to the same as previous.

For the runtime cost in Eq. (3), the first term, $\frac{M}{p}2^n\tau_0$, is the random sampling cost incurred by the manager in database construction, which occurs $\frac{M}{p}$ times in the shared database case. The second term, $2^n\sum_{k=1::p}^{M} k\tau_s$, is the parent search

cost, which is identical to term 2 of Eq. (1). The third term, $M2^{n+1}\tau_v$, is the message passing cost of sending each worker a database (for a total of M passes). The fourth term, $\frac{M}{p}C_Q$, is the cost of applying QAA and is distributed at runtime. The last term, $M\tau_v$, is the cost of sending all M found nodes back to the manager. To attain the total cost in Eq. (4), the distributed term of Eq. (3) (term 4) is multiplied by p to find sequential runtime cost.

There are two main cost differences between the unshared and shared database formulations in creating an M-node tree. In both the runtime cost, in Eq. (1), and the total cost, in Eq. (2), the unshared algorithm incurs a message passing cost of $p\sum_{k=1::p}^{M} k\tau_v$ in passing the growing tree (from 1 to M nodes) to p workers. In Eq. (3) and (4), the shared algorithm incurs a message passing cost of $M2^{n+1}\tau_v$ instead, in passing the database to workers (M times). The other terms in the runtime costs between the versions are identical. Whether the difference in message passing costs favors one scheme over the other is dependent on the relative sizes of M and 2^n. For small trees and large databases, the unshared database method incurs a smaller cost. For large trees and small databases, the shared database method incurs a smaller cost.

The second difference is in the total cost. In Eq. (2), the unshared algorithm incurs a total database creation cost of $M2^n\tau_0 + 2^n p\sum_{k=1::p}^{M} k\tau_s$ to create an M node tree. In Eq. (4), the shared algorithm incurs a total database creation cost of $\frac{M}{p}2^n\tau_0 + 2^n\sum_{k=1::p}^{M} k\tau_s$, effectively giving a p-fold reduction in total database creation cost. We refer to this as *increased database efficiency*, since an M-node tree is created with fewer database constructions.

In Fig. 3, we compare runtime costs of the two architectures when weights are assigned to each cost. As noted above, at runtime, the cost difference comes in message passing either the tree or the database, so the ratio of number of desired nodes M to total database size is plotted against cost. We show cost curves for the 8 and 32 core cases for both architectures to further understanding on how costs scale with number of cores. As is expected, 32-core cases minimize costs across the entire plotted domain for both architectures. Additional cores seem to minimize the unshared database cost to a greater extent than the shared database cost. At small tree sizes relative to the database, the unshared architecture minimizes cost, and the curves appear to converge as the end tree size eclipses the database. The visualization is possible through the log scaling of the cost, but this scaling obscures the fact that the shared formulation actually becomes lower cost at high ratios of $\frac{M}{2^n}$.

In Fig. 4, we depict a semilog plot of the cost differential between the shared and unshared architectures over the same domain of ratios of tree to database size. For smaller final trees, the unshared formulation maintains a cost advantage, but the advantage skews strongly to the shared database as the tree size grows. This effect is obscured in Fig. 3 by the loglog nature of the scaling and apparent equal performance on the upper end of the domain. The cost differential over the entire

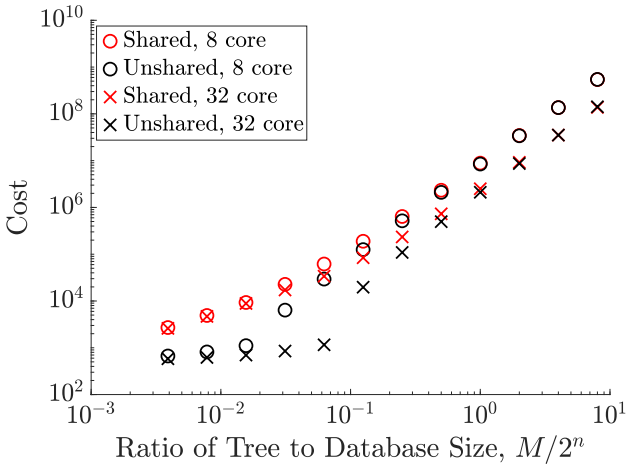

FIGURE 3. A comparison in loglog space of the cost differences between the unshared and shared database architectures as the relative final tree size M varies with respect to database size 2^n.

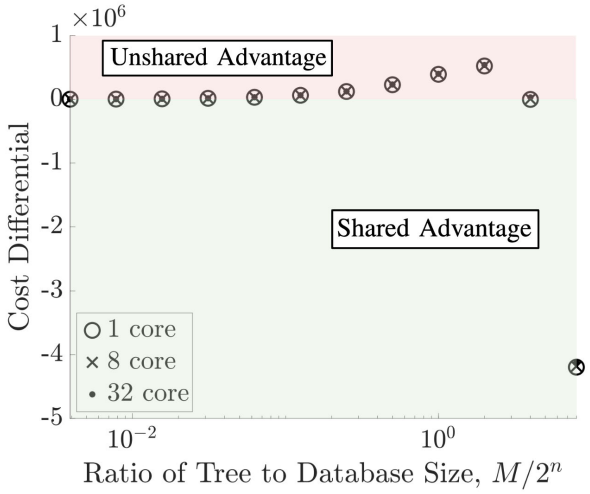

FIGURE 4. The differential cost between the shared and unshared database architectures as the relative final tree size M varies with respect to database size 2^n.

domain appears relatively similar between 1, 8, and 32 core cases.

A key assumption in the above analysis is that for both algorithm architectures, in each iteration, the p cores add p nodes to the tree. For both architectures this is overlooking the fact that the database cannot be optimally amplified, as applying an operator can only be done an integer number of times, and the optimal amplification number is exactly defined using an irrational number (π). Furthermore, even if optimal amplification was possible, in general there is still a nonzero probability of measuring a 'bad' (unamplified; as defined by the oracle) element, which should not be admitted to the tree (and can be caught with a final deterministic check). This simple fact eliminates the possibility that in each iteration, the p cores can always add p nodes to the tree.

A further complication for the shared database setup is that nodes may find identical solutions, further lowering efficiency. In the following section we explore to what extent this is likely, and guidelines for minimizing this effect to allow high database use efficiency.

A different possible solution is to use a partitioned "shared" database, where a single larger database is passed to each worker with a partition rule such that each worker searches different and non-overlapping portions of the database. This would eliminate the repeated solution problem, but is more database efficient (if each partitioned database is considered part of one database), but would require larger databases and is functionally equivalent to the unshared database architecture, with the manager creating databases rather than the workers. A larger database with more solutions also helps to eliminate the repeated solution problem. We also note that although classically connection costs $\tau_Q \gg \tau_{0,s,\nu}$, this may no longer be the case when quantum computers are used to determine reachability.

D. PQ-RRT PROBABILITY RESULTS

In this section, we characterize p parallel quantum workers finding multiple solutions when Pq-RRT is operating in a shared database setup. Since all the workers are independently analyzing the same 2^n-sized database D, with m oracle-marked solutions, in general multiple workers may arrive at the same solution. This represents an efficiency loss to the shared database setup, so in what follows we characterize the worst and best case events. The worst case event is all workers arriving at the same solution, which has the runtime performance as non-parallel q-RRT but is p-times less oracle call efficient. The best case event is each worker finding a different solution, which has no runtime or oracle call efficiency loss, and ends with p solutions. Only when $p \geq m$ can all solutions be found in a single parallel pass. To build fast solutions, an understanding of the effects of choices of p (and to some extent m) is necessary to maximize efficiency.

We assume that the database is optimally amplified according to $i_{\max}$ applications of Q, where $i_{\max}$ is given by,

$$i_{\max} = \frac{\pi}{4}\sqrt{2^n/m}\,;$$

see [40]. We note that our connectivity analysis on estimating database correctness in [1] can be applied to the section at hand in order to attain optimal amplification (to maximize chances of measuring a solution). In what follows, let G be the event that a good state, as defined by the oracle, is measured by a worker after $i_{\max}$ iterations of Q.

1) WITHOUT ORACLE ERRORS

Lemma 1: Let there be a parallel quantum process with p workers and a shared 2^n-sized database with m solutions. The probability that all p workers find the same solution is,

$$\mathbb{P}(\textit{same solution}) = \mathbb{P}(G)^p m^{1-p}, \tag{5}$$

where $\mathbb{P}(G)$ is the total probability of event G,

$$\mathbb{P}(G) = \sin^2((2i_{\max}+1)\theta),$$

and where θ is defined such that $\sin^2(\theta) = \frac{m}{2^n}$.

Proof: To attain this result, we observe that after $i_{\max}$ iterations of Q, all m solutions have equal probability of measurement given by $\mathbb{P}(G)/m$. The probability that all p workers measure a *particular* solution i is,

$$\mathbb{P}(\text{particular solution}) = \left(\frac{\mathbb{P}(G)}{m}\right)^p,$$

and this is multiplied by m to generalize to finding *any* same solution, yielding Eq. (5). The total good measurement probability and the definition of θ can be found at [38]. □

Lemma 2: For $m \geq p$ and $m, p \in \mathbb{N}$, the probability that all workers find different solutions is given by,

$$\mathbb{P}(\textit{different solutions}) = \frac{\mathbb{P}(G)^p m!}{m^p (m-p)!}, \tag{6}$$

Proof: This result follows from m permute p (the number of possible ways p objects can be selected, without replacement, from m possibilities) over the total number of possible outcomes m^p. This is scaled by the likelihood that all workers find a correct solution, $\mathbb{P}(G)^p$, to yield Eq. (6). □

For the worst case scenario in Eq. 5, as $m \to \infty$, $\mathbb{P}(\text{same solution}) \to 0$. This makes intuitive sense: as the number of available solutions increases, the likelihood of all workers finding the same solution decreases as a function with power $1-p$, where $p \geq 2$, as it is only sensible to conjecture about the parallel behavior of 2 or more workers.

For the best case scenario in Eq. 6, as $m \to \infty$, $\mathbb{P}(\text{different solutions}) \to \mathbb{P}(G)^p$. This also makes intuitive sense: as the number of solutions increases, the likelihood of all workers finding different solutions approaches the total likelihood of all workers finding a solution. We note that for $m \geq p$ and $m, p \in \mathbb{N}$:

$$\lim_{m\to\infty} \frac{m!}{m^p (m-p)!} = 1.$$

The number of solutions m should be as large as practicable to reduce efficiency loss through oracle overlap.

Lemma 3: For $p \geq m$, the expected number of workers p, to find all m solutions within D in one pass, is given by,

$$E(p) = \frac{mH_m}{\mathbb{P}(G)},$$

where H_m is the m^{th} harmonic number. Equivalently, $\frac{E(p)}{p_2}$ also describes the expected number of passes at a single database that the set number of workers p_2 must make to find all solutions.

Proof: We reach this result from the application of the Coupon Collector's problem [41], with a minor modification, to the independent quantum computing worker processes. Briefly, the coupon collector's problem concerns questions about the "collect all coupons from cereal boxes to win" contest. In this context, solutions in the database represent

coupons (to be found with a certain probability), and number of workers represents (the expectation of) how many cereal boxes must be opened to find one of each solution/coupon. The result takes into account that workers may return the same solution. This application is scaled by the total probability of correct solutions, $\mathbb{P}(G)$, to account for the proportion of the time when a good solution is not measured. □

Lemma 3 allows a parallel (and repeated) architecture to be chosen based upon knowledge of m, such as from our connectivity analysis on estimating database correctness in [1]. Additionally, this leads to the database construction tool described in the following section that allows the proportion $m/2^n$ to be made larger or smaller. We remark that in general, it is possible to calculate the probability of n workers coinciding on the same exact solution, as it relates to the multinomial distribution.

2) WITH ORACLE ERRORS

We also consider the case where the oracle is making repeated false positive and false negative errors. Let the probability that a state is marked by the oracle incorrectly as good be given by $q \in [0, 1]$ (false positive), and let the probability that a state is marked by the oracle incorrectly as bad be given by $v \in [0, 1]$ (false negative). Let the number of ground-truth solutions in the database tagged by the oracle as a solution be $m_1 \leq m$. Let the number of actual ground truth solutions in the database, which were mistakenly tagged by the oracle as bad be m_2, such that,

$$q = \frac{1 - m_1}{m}, \ v = \frac{m_2}{2^n - m},$$

as shown in Fig. 5.

First, we derive the likelihood that a single worker finds a real solution. Let G^* be the event that a real, ground truth solution (not according to the oracle) is measured for addition to the tree after optimal amplification with QAA.

Lemma 4: The total probability of measuring real, ground truth solutions is,

$$\mathbb{P}(G^*) = \frac{m_1}{m}\mathbb{P}(G) + \frac{m_2}{2^n - m}(1 - \mathbb{P}(G)).$$

Proof: We attain this result by adding the probability of measuring a correctly tagged good solution, $\frac{m_1}{m}\mathbb{P}(G)$ to the probability of measuring an incorrectly tagged bad solution, $\frac{m_2}{2^n - m}(1 - \mathbb{P}(G))$. □

The following lemma is a modification of Lemma 1 to include oracle errors.

Lemma 5: The probability that all p workers find the same ground truth solution, adjusted for oracle mistakes, is,

$$\mathbb{P}(\textit{same solution}) = m_1 \left(\frac{\mathbb{P}(G)}{m}\right)^p + m_2 \left(\frac{1 - \mathbb{P}(G)}{2^n - m}\right)^p. \tag{7}$$

Proof: This result follows from the addition of the probability of all p workers measuring a *particular* correctly tagged good solution, $\left(\frac{\mathbb{P}(G)}{m}\right)^p$, and the probability of all p workers measuring a *particular* incorrectly tagged bad solution, $\left(\frac{1-\mathbb{P}(G)}{2^n - m}\right)^p$. Each of these is multiplied by m_1 and m_2 respectively, to consider *any* real solution, then added together to yield Eq. (7). □

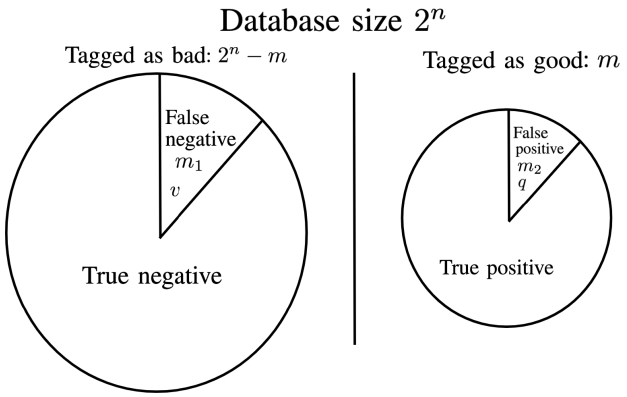

FIGURE 5. A graphical depiction of the false positive and false negative regions of a database with good and bad tags by an oracle. Of the m good tags, m_1 are true good tags, with a false positive probability of q. Of the $2^n - m$ bad tags, m_2 are actually good elements, with a false negative probability of v. After optimal QAA, the probability of a tagged-as-good state being measured is $\mathbb{P}(G)$.

For the worst case scenario in Lemma 5, again, as $m \to \infty$, $\mathbb{P}(\text{same solution}) \to 0$. The following lemma is a modification of Lemma 3, adjusted for oracle false positive errors.

Lemma 6: The expected number of workers p to find all m_1 ground truth solutions, when false positive oracle errors are taken into account, is given by,

$$E(p^*) = \frac{m_1 H_{m_1}}{\frac{m_1}{m}\mathbb{P}(G)},$$

where H_{m_1} is the $m_1{}^{th}$ harmonic number.

Proof: This result follows similarly to Lemma 3, with m_1 rather than m, and where the expectation from the Coupon Collector's Problem is scaled by the proportion of the time that a ground truth good solution is found, $\frac{m_1}{m}\mathbb{P}(G)$. □

Remark 4: Oracle false negative errors, when considered in Lemma 6, transform the problem into a simple version of the Weighted Coupon Collector's Problem [42] (also known as McDonald's Monopoly), with some "rare coupons" to find (represented by the m_2 false negative solutions), and some "common coupons" (represented by the m_1 real positive solutions). In reality, after optimal amplification, the probability of measuring the m_2 good solutions incorrectly tagged as bad, $\frac{m_2}{2^n - m}(1 - \mathbb{P}(G))$, is small, and additionally we care more about finding correctly identified good solutions than determining incorrectly tagged bad solutions from a database.

E. QUANTUM DATABASE ANNEALING

In this section, we define the Quantum Database Annealing (QDA) strategy, shown in Alg. 3. QDA builds databases with

elements constrained to a certain distance from the parent node, as defined by a temperature matrix H and iterator h. The QDA strategy is an alternative to standard database construction and is inspired by the optimization technique simulated annealing and our investigations into oracle call constraints in [1]. It represents a possible way to guide database construction to achieve a particular algorithmic goal, such as approximately selecting m (with regard to the previous section), or in this case, initial fast expansion via spread node placement followed by increasing density through closer node placement.

In a broad sense, sampling strategies for motion planning have been explored since the beginning, with strategies such as medial axis sampling, boundary sampling, Gaussian (obstacle) sampling, goal biasing, and hybrid schemes [21]. QDA is distinct from current classical computing approaches because the initial goal in this quantum formulation is to make sample connections less likely. When paired with a large database, QDA exploits quantum computing's ability to quickly find unlikely solutions, resulting in a motion planner that can explore very quickly when measured on oracle calls.

QDA first samples according to a uniform distribution over the configuration space C. The nearest (Euclidean distance) existing node to the sample is chosen as the parent. Next, the resulting parent-child relationship is constrained to be within a ball of radius $H(h)$ (with iterator h) while maintaining child-sample direction. The resultant pair is added to the database. An alternative is to sample initially over a disc or boundary at a distance constrained by $H(h)$.

In the beginning, with high temperature (when $H(h)$ is large), QDA will build a large database of further away and therefore less likely solutions. This allows further reachable solutions to be found quickly as compared to q-RRT and RRT. As the path planning problem continues (as h increases), the temperature ($H(h)$) may drop to account for the addition of new nodes and to increase the ratio of good solutions in the database. The database size 2^n may also drop throughout the problem to increase efficiency, as when there are more solutions, smaller databases function as well as larger. When no additional information is known to guide sampling region, an alternative but similar sampling method in very large bounded configuration spaces is to build extremely large databases of unlikely solutions in an attempt to span long obstacle free channels quickly.

The differences between q-RRT and q-RRT with QDA, Alg. 3, lie in the latter algorithm's lines 2, 7, and 21. Alg. 3 line 2 is where the temperature array H is defined and iterator h initialized. On Alg. 3 line 7, the temperature constraint is carried out by modifying the random point t with respect to P, the closest parent of t in T. On Alg. 3 line 21, the iterator h is incremented to allow different temperatures on future database constructions. In the defined formalism, the database size 2^n is set and not decreased.

This approach to guided sampling is distinct from efforts within guided sampling in non-quantum literature. Generally, the point of guided sampling is to make the search for additional states and solutions easier. Additionally, there is no reason (until the introduction of quantum computers to motion planning) to construct *databases* of possible solutions, the algorithms always progress by testing one possible state at a time. With QDA, however, we design a guided sampling scheme with a polar opposite goal to existing efforts: making the solution process more difficult. This is because we are able to exploit the quantum advantage, and by making the solution process more difficult, we are able to find 'better' solutions in the sense of solutions that allow vigorous (and provable with respect to metrics such as reachability and safety) expansion into an environment. When constructing databases, we also think of guided sampling in a different lens.

Algorithm 3 q-RRT with Quantum Database Annealing

Input: $x_0, x_G\ n,$ oracle $\mathcal{X}$
Output: Path γ

1: Init tree T with root at x_0
2: Define temperature array H, index $h = 0$
3: **while** $x_G \notin T$ **do**
4: **for** $i = 1$ to 2^n **do**
5: $t =$ random point
6: $P =$ closest parent of t in T
7: Constrain t to disc of dist. $H(h)$ from P
8: $D(i) = [t;\ P]$
9: **end for**
10: Enumerate D via $F : \{0, 1\}^n \rightarrow D$
11: Init n qubit register $|z\rangle \leftarrow |0\rangle^{\otimes n}$
12: $|\Psi\rangle \leftarrow \mathbf{W}\,|z\rangle$
13: **for** $i = 1$ to 2 **do**
14: $|\Psi\rangle \leftarrow Q(\mathcal{X})\,|\Psi\rangle$
15: **end for**
16: $[x_{add}, P] \leftarrow F(\text{measure}(|\Psi\rangle))$
17: **if** $\|x_{add} - x_G\| < \delta$ **then**
18: $x_{add} = x_G$
19: **end if**
20: Add $[x_{add}, P]$ to T
21: $h++$
22: **end while**
23: Return path γ from T

IV. RESULTS AND DISCUSSION

In this section, we show tree creation comparison results within two dimensional obstacle environments for Pq-RRT, q-RRT, Parallel RRT, and RRT. Direct comparisons highlighting the simulated quadratic runtime advantage of q-RRT over standard RRT are shown in detail in our previous work at [1]. Unless otherwise stated, results are presented comparing algorithm performance for solving the same problem in the same randomized obstacle environments. Both Pq-RRT and Parallel RRT are implemented with eight cores (workers). Both quantum algorithms use databases of size 2^8, and the classical versions of the algorithms (RRT and Parallel

RRT) replace the database construction and quantum search process with single reachability tests. The specific version of Parallel RRT is a Manager-Worker formulation (outlined in [26] under Manager-Worker RRT), where a manager processes the tree and assigns single-node expansion work to workers, as expansion is the computationally expensive part of planning.

All path planning simulations are run with Matlab v2022b on an eight core MacBook Pro with M2 chip. Quantum states and algorithms are simulated with the Matlab Quantum Computing Functions library [40]. A selection of Matlab code is available at github.com/pdlathrop/PQRRT. This implementation was chosen over IBM's Qiskit and over actual experimental quantum devices in order to: maintain comparability with our previous work regarding q-RRT, enable full back-end probability amplitude control for verification and testing, allow integration with classical computing functions and parallel structures, and permit relatively large qubits and long runtimes. All algorithms use the following arbitrary dynamics and reference tracking controller to test reachability for node admittance to the tree,

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t)\ ,\ x(0) = x_{\text{parent}}, \\ A &= \begin{bmatrix} -1.5 & -2 \\ 1 & 3 \end{bmatrix},\ B = \begin{bmatrix} 0.5 & 0.25 \\ 0 & 1 \end{bmatrix}, \\ u(t) &= -Kx(t), \\ K &= \begin{bmatrix} 1.9 & -7.5 \\ 1 & 7 \end{bmatrix}. \end{aligned}$$

The constant gain matrix K can be any matrix such that the closed loop system is stable.

A. NODE PLACEMENT AND ORACLE CALLS

In this section, we highlight the advantages and disadvantages of q-RRT and Pq-RRT over RRT and Parallel RRT in being able to add nodes to the tree. We show performance compared to runtime in seconds, which we call wall-clock time to highlight that this is the "real" runtime of the simulations, and then to number of oracle calls, which functions as the projected runtime improvement if algorithms are run on quantum devices. For performance metrics, we consider two quantities: number of oracle calls and number of nodes. Number of oracle calls is the metric for comparing how much Parallel q-RRT is able to speed up computation as compared to q-RRT. For all other comparisons, number of nodes is the chosen metric, as it signifies on a functional level the ability of each algorithm to search for a solution. Each point in Fig. 6- 9 represents one 30-node tree creation, chosen to showcase average performance.

First, in Fig. 6 we show the wall-clock speed of Pq-RRT and q-RRT in being able to perform oracle calls to analyze the amount of computation speedup achieved. Next, we compare the wall clock speed of the two classical algorithms (in Fig. 7) and the two quantum algorithms (in Fig. 8) to study the relative performance gain in parallelizing the quantum routines compared to classical parallel advantage. Lastly,

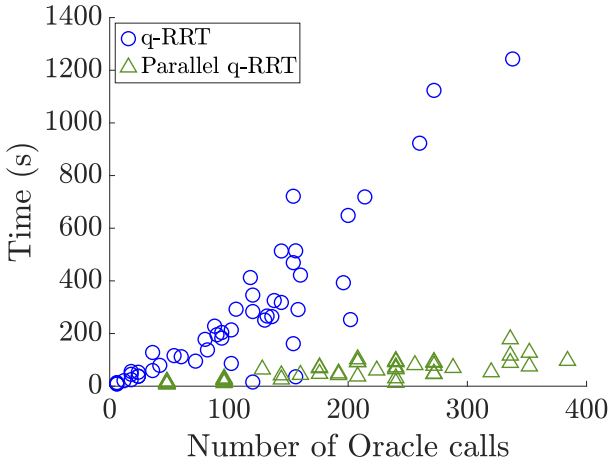

FIGURE 6. Comparison of the wall-clock speed (in seconds) of q-RRT, and shared database Pq-RRT in performing oracle calls or reachability tests.

in Fig. 9, we change to an oracle call (our quantum time surrogate) vs node creation comparison to show the efficiency of the quantum algorithms in admitting nodes to the tree. The performance advantage of Pq-RRT is inferred to combine the advantage shown over q-RRT in Fig. 8 and the advantage shown over the classical algorithms shown in Fig. 9.

Figure 6 depicts the wall-clock speed of q-RRT and parallel q-RRT in performing oracle calls. Pq-RRT performs oracle calls in less time than q-RRT, as it more efficiently uses multiple workers to retrieve information from created databases. The relationship between oracle calls and time is approximately linear (as expected), and with a linear fit (using linear least squares), Pq-RRT shows a smaller slope (0.20) compared to q-RRT (3.52), with slope referring to seconds per oracle call. A lower (or shallower) slope is more efficient, as for a fixed number of nodes, a lower time (y-value) corresponds with increased efficiency. A classical computing shortcut is used which allows Pq-RRT to be 17.6 times more efficient in performing work. In quantum computing simulation, a single created database is analyzed for reachability once, then amplified once, and each worker then measures a solution. This shortcut would not be possible on a quantum device, as qubits, once they are created, cannot be copied, so each worker would need to perform the reachability analysis separately. This would change the expected slope difference to be approximately 8 times less than 17.6 (as 8 cores are used), for a total work (oracle call) efficiency gain of 2.2.

Figure 7 depicts the wall-clock speed of RRT and Parallel RRT in admitting nodes to the graph, as opposed to performing oracle calls. The same data for q-RRT and Pq-RRT is shown in Fig. 8. This comparison factors in differing node-admission oracle call efficiencies. The parallel versions of both algorithms, Parallel RRT and Pq-RRT, each are on average more time efficient than the non-parallel versions in admitting reachable states to the tree.

The intuition behind slope in Figures 7 and 8 is seconds per node, with a shallower slope meaning more efficient.

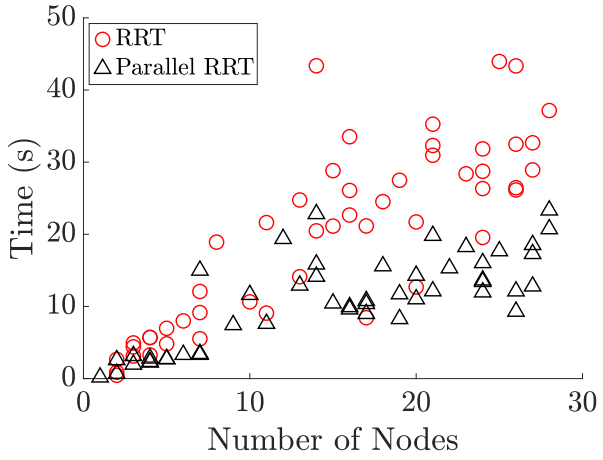

FIGURE 7. Comparison of the wall-clock speed (in seconds) of RRT and parallel RRT in admitting reachable states to the tree.

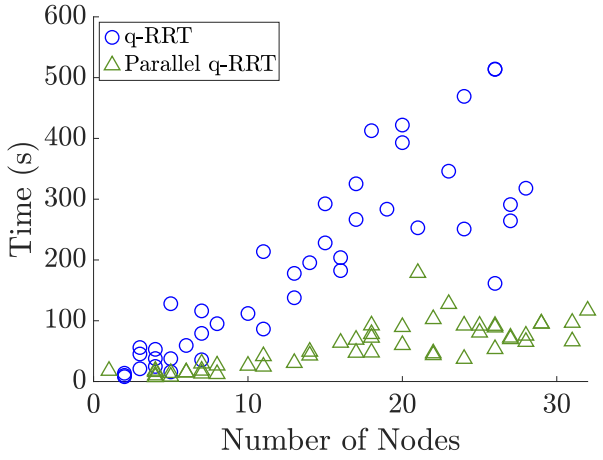

FIGURE 8. Comparison of the wall-clock speed (in seconds) of q-RRT and Pq-RRT in admitting reachable states to the tree.

Pq-RRT (slope 3.17) in particular shows greater improvement over q-RRT (slope 25.3) than Parallel RRT (slope 0.58) over RRT (slope 1.23), as is evidenced by a larger difference in slope (8.0-fold efficiency increase compared to 2.1-fold), as calculated with a linear fit and linear least squares. The quantum algorithms, when measured by real time, lag behind both non-quantum RRT versions because they are not benchmarked on quantum computers (see the *y*-axis label differences between Fig. 7 and Fig. 8). The quantum computing simulations are performed via large arrays on classical devices. On a quantum device, we expect the run-time to be analogous to oracle calls, as discussed next.

Figure 9 depicts the oracle call efficiency of all four algorithms in admitting reachable states to the tree as a function of the number of oracle calls it takes. This figure is analogous to expected run-time when the quantum algorithms are executed on a quantum device. Slopes are found with a linear fit using linear least squares and represent the number of oracle calls per node, again with a shallower slope meaning more efficient. The efficiency advantage of q-RRT (slope

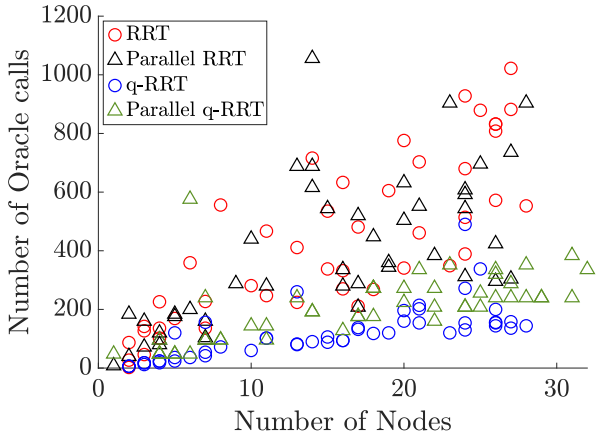

FIGURE 9. Comparison of the oracle call efficiency of RRT, parallel RRT, q-RRT, and Pq-RRT in admitting reachable states to the tree.

7.6) and Pq-RRT (slope 10.7) in admitting reachable states is shown over RRT (slope 21.5) and Parallel RRT (slope 19.4). The q-RRT algorithm is more efficient than Pq-RRT due to the fact that multiple workers can simultaneously return the same solution from a database (as explored in Props. 1 to 6), and repeat solutions are discarded. However, Pq-RRT is capable of making simultaneous oracle calls with different workers, so for parallel vs not parallel time comparisons we refer the reader to Fig. 7 and Fig. 8.

The conclusions of the above analysis are the following: Pq-RRT is more time-efficient than q-RRT in performing work and placing nodes, Pq-RRT shows a greater time efficiency increase over q-RRT than Parallel RRT does over RRT, and q-RRT is slightly more oracle-call-efficient than Pq-RRT, but both quantum algorithms are more oracle-call-efficient than the classical algorithms.

B. EXPLORATION SPEED

The results of this section are extensions to our results in [1] between q-RRT and RRT, showing q-RRT's ability to explore quickly and in a generalized environment. We show a heat-map of state space nodes placed within a certain number of oracle calls. Oracle calls are chosen as a substitute to time because the quantum computer simulation performs slowly on classical devices. Actual runtime is expected to be analogous to the number of oracle calls, as reachability tests for the local planner consume the majority of the algorithm runtimes. Each algorithm is tested over 100 trials. Each trial is cut off after a certain number of oracle calls to show each algorithm's speed of node placement. Let oracle efficiency be the ratio of total nodes placed over total oracle calls. In each figure, the red circle refers to a goal zone.

Figure 10 depicts the initial exploration speeds, from 0 to 10 oracle calls, of RRT and q-RRT. Each path planning problem is cut off after 10 oracle calls and a heatmap is created of the total node placement in the state space over 100 trials. The q-RRT method shows much faster initial node placements over RRT, admitting 372 nodes with an

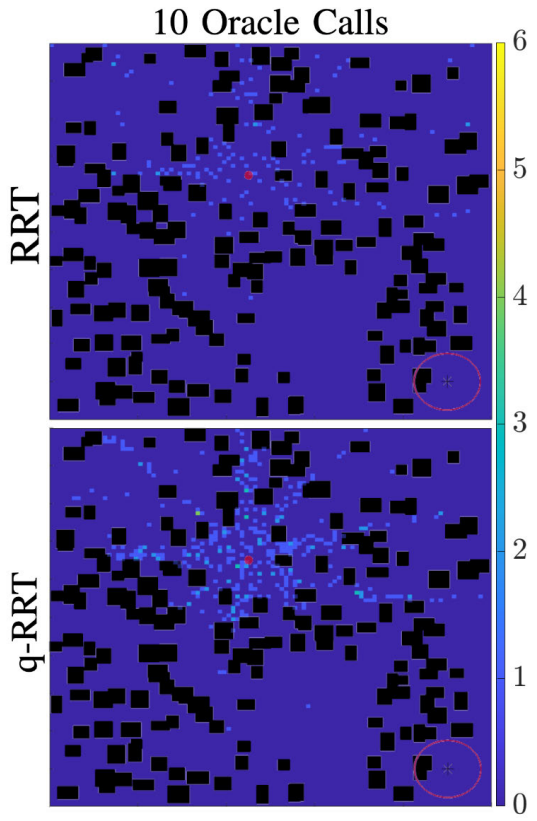

FIGURE 10. Comparison of initial exploration speeds (up to 10 oracle calls) of RRT and q-RRT. Data is shown as a state space heat-map of node placements over 100 trials of each algorithm in the shown obstacle environment. A goal zone is shown as a red ring in the bottom right, and the heatmap color key is shown on the right of the graph.

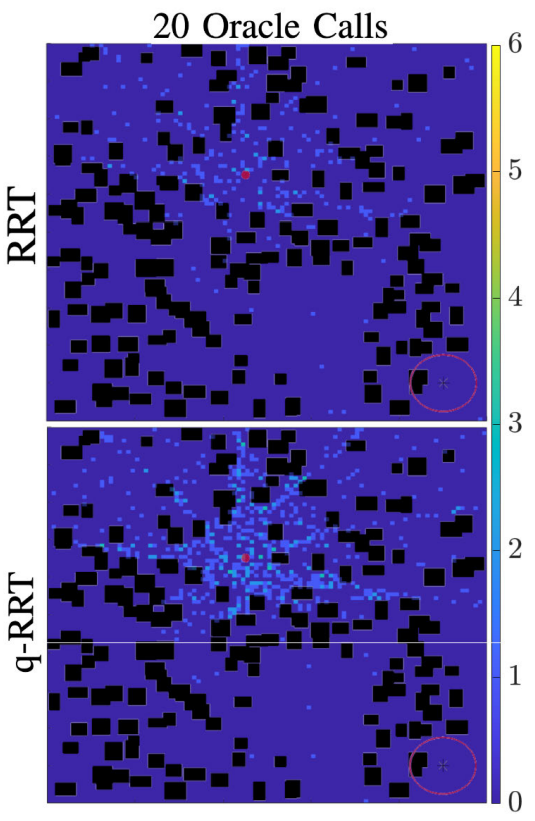

FIGURE 11. Comparison of middle-time exploration speeds (up to 20 oracle calls) of RRT and q-RRT. Data is shown as a state space heat-map of node placements over 100 trials of each algorithm in the shown obstacle environment.

oracle efficiency of 31.2%. The q-RRT method has more than a thousand total oracle calls due to the inclusion of a finalizer line before nodes are admitted to the tree. RRT admitted 125 nodes with an oracle efficiency of 12.5%. Node placement is more dense both in the initial node pocket and along lines exploring outward between obstacles away from the initial node.

Figure 11 depicts the middle-time exploration speed, from 0 to 20 oracle calls, of RRT and q-RRT. Similarly, each path planning problem is cut off after 20 oracle calls and a heatmap created from the total node placement of each algorithm over 100 trials. The q-RRT method shows much faster and more full middle-time node placement, admitting 650 nodes with an oracle efficiency of 31.0%. RRT admitted 231 nodes with an oracle efficiency of 11.6%. Node placement is more "full" in the initial pocket, and is much more dense along lines exploring out between obstacles from the initial node.

Figure 12 depicts the "late-time" exploration speed, from 0 to 40 oracle calls, of RRT and q-RRT. Heatmap creation is again similar to previous. The q-RRT method has admitted more nodes in nearby pockets, and has a more dense spread of nodes in further away regions, admitting 1091 nodes with an oracle efficiency of 26.0%, compared to RRT, which admitted 526 nodes with an oracle efficiency of 13.2%. The average oracle efficiency of q-RRT has dropped somewhat compared to the initial and middle time exploration, and this is due to the fact that, as the existing tree grows, new random points are more likely to be reachable to the existing graph. This serves to allow RRT to catch up in terms of efficiency, and q-RRT's created database, on average, has allowed more solutions. The quantum version of the algorithms thrive (in comparison) in situations where there are few solutions. Heatmaps for Pq-RRT were also created for 10, 20, and 40 oracle call cases as shown in Fig. 10- 12. The Pq-RRT heatmaps were omitted, as results were largely similar between q-RRT and Pq-RRT. This is similar to findings in Fig. 9 that, when compared over oracle calls (quantum time surrogate), Pq-RRT's advantage is not apparent, as Pq-RRT is able to make simultaneous oracle calls.

C. NARROW CORRIDOR EXPLORATION

The ability of motion planning algorithms to find paths through narrow corridor environments serves as a benchmark for the ability to find difficult solutions in narrow spaces. In Figs. 13 and 14 we show, through a heatmap, the ability of q-RRT to find passage through a narrow corridor when compared to RRT. The figures depict a heatmap of node placements of 50 trials of each algorithm in the overlaid

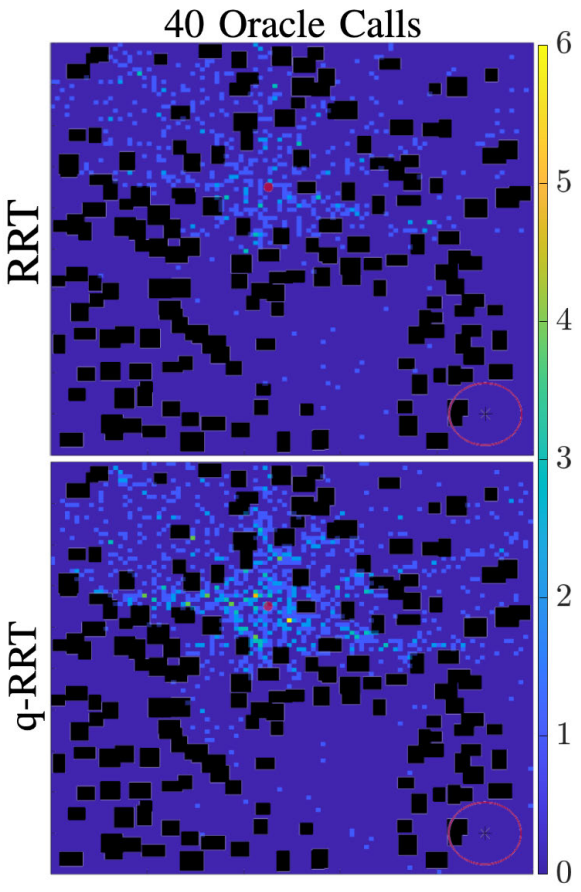

FIGURE 12. Comparison of late-time exploration speeds (up to 60 oracle calls) of RRT and q-RRT. Data is shown as a state space heat-map of node placements over 100 trials of each algorithm in the shown obstacle environment.

environment, where each method is cut off after 25 oracle calls to analyze ability to quickly place nodes in the narrow corridor. Obstacles are depicted in black, and are distributed randomly on both sides of the narrow corridor, which is created by 2 large obstacles.

The q-RRT method placed 47 nodes in the narrow corridor, compared to RRT's 14 nodes. Results are presented with no guided sampling or known-goal direction to guide sampling. The q-RRT algorithm is quicker to find paths into narrow corridors toward possible goal locations.

D. QUANTUM DATABASE ANNEALING

We compare the abilities of Quantum Database Annealing and standard q-RRT database construction to create trees that spread across larger configuration spaces with a large number (6025) of obstacles. In this formulation, Quantum Database Annealing is initially creating databases of points at a distance between 2.7 and 4.2 units from current nodes, then dropping that range to between 0.8 and 2.0 units to fill in the space around the initially spread tree. On the other hand, q-RRT with standard database construction is sampling across the entire configuration space C. Both algorithms are

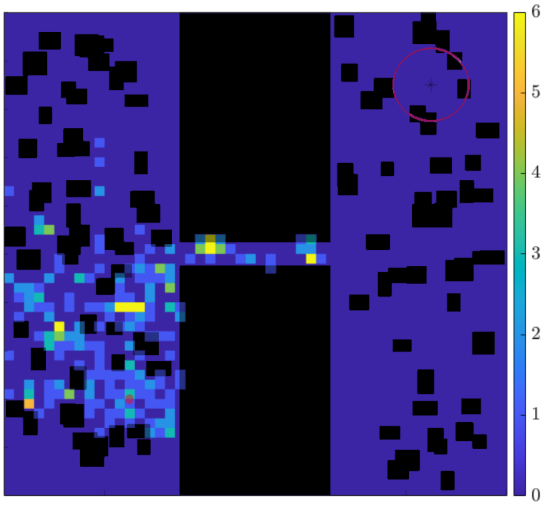

FIGURE 13. A heatmap of q-RRT's node placement in a narrow corridor environment (up to 25 oracle calls) over 50 trial runs, with 47 nodes in the channel. Obstacles are depicted in black, and a color key of node placements is shown to the right of the graph.

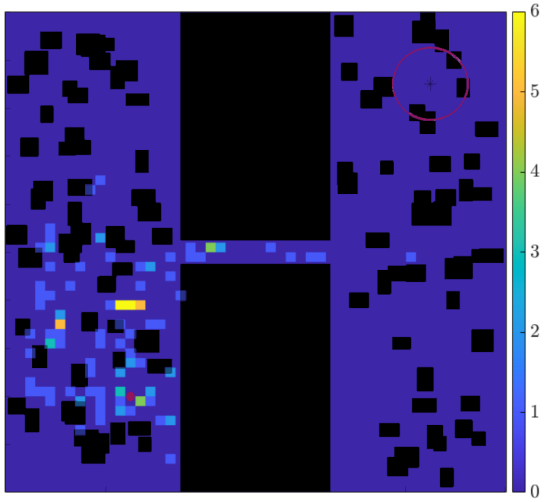

FIGURE 14. A heatmap of RRT's node placement in a narrow corridor environment (up to 25 oracle calls) over 50 trial runs, with 14 nodes in the channel.

using databases of size 2^9. Fig. 15 depicts a 16-node tree with initial fast expansion made with Quantum Database Annealing, and Fig. 16 shows continued node addition to a 48-node tree with lower temperature to fill in the area around the initial spread tree. Fig. 17 depicts the standard q-RRT created 16-node tree in the same environment, to compare against Fig. 15. Obstacles are depicted as small black rectangles, the root node of each tree is shown as a black circle, nodes in each tree are shown as red circles, and parent child connections are shown as black lines.

The resulting trees differ in how spread they are for the same number of oracle calls (the quantum analog of runtime). Quantum Database Annealing initially creates nodes an average of 3.68 units away (with the above temperature

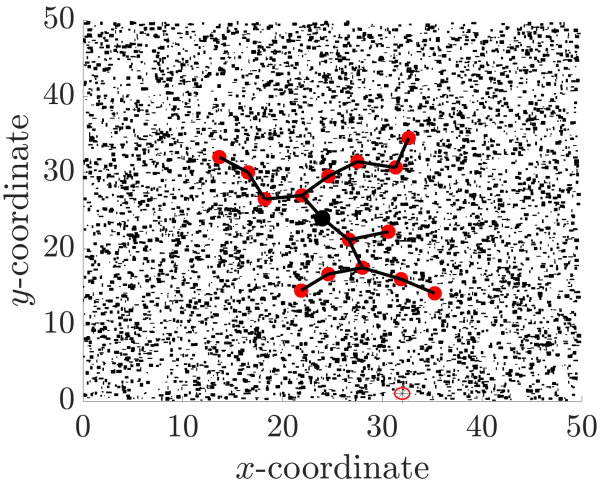

FIGURE 15. A 16 node tree created with quantum database annealing with high temperature, showing fast initial exploration. The root node is a black circle and tree nodes are red circles. Parent-child relationships are shown via black lines, and 6025 obstacles are depicted as small black rectangles.

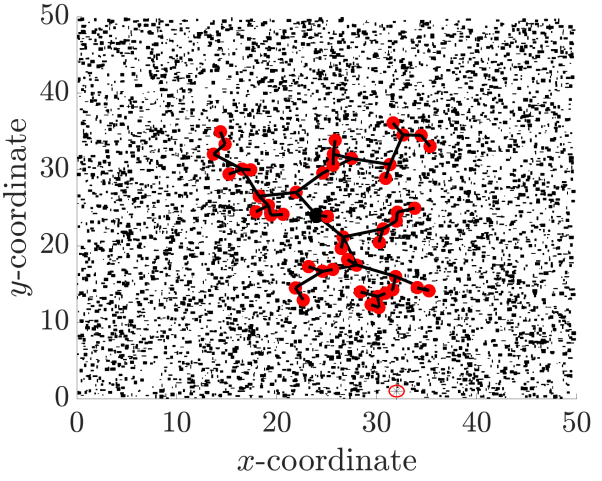

FIGURE 16. A 48 node tree created with quantum database annealing with initial high temperature, then a dropping temperature, showing how temperature can be used to fill in the area around a spread tree. The root node is a black circle and tree nodes are red circles. Parent-child relationships are shown via black lines, and 6025 obstacles are depicted as small black rectangles.

setting) from their parent and standard q-RRT creates nodes an average of 1.70 units away from their parent. For a fixed number and size of obstacles, it should be noted that sampling parameters affect the average distance in the q-RRT tree, and different average distances can be obtained by varying the size of the database. For uniform sampling over C, as the database becomes larger, the average distance drops, as nodes are more frequently found near existing nodes. For equal-sized large databases, in the same amount of (quantum) time, QDA is able to create trees with more spread, as only further away nodes are admitted to the database. The temperature construct allows a balance between exploration and density of nodes, enabling a version of q-RRT that can connect distant regions of a configuration space very quickly before back-filling with lower temperature.

The proposed formulation combines QDA with q-RRT (in replacement of standard uniform database construction) in an effort to show comparisons with the most basal algorithm possible. Both Pq-RRT and QDA are strategies to increase the efficiency of q-RRT, but in different ways, and compared over different metrics. We expect the strategies to be complementary because while Pq-RRT increases speed through decreasing seconds per node, QDA increases speed through allowing higher-quality and more exploratory nodes themselves. In this way, a parallel formulation may allow faster node placement, and QDA may simultaneously allow the placed nodes to explore more quickly. Optimal integration to consider non-uniform temperatures between parallel threads, message passing issues, and possible spatial database decomposition represents a further research direction. We reserve showing complementary behavior and modifying QDA to allow additional parallel synergy for future work.

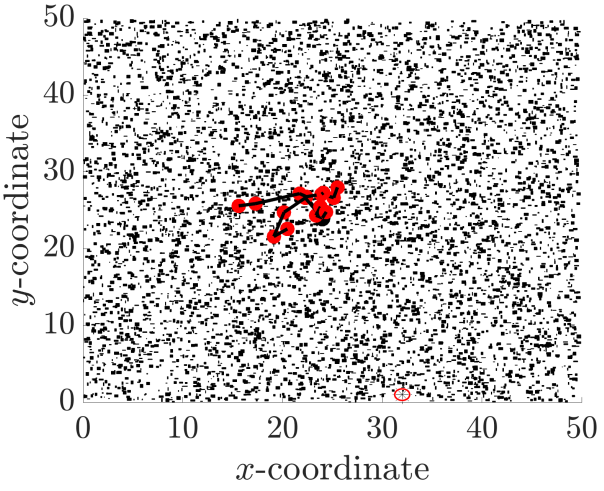

FIGURE 17. A 16 node tree created with q-RRT (with standard database construction) in the same 6025 obstacles environment.

V. CONCLUSION

To generalize and extend q-RRT, we provide analysis in more general obstacle environments, a formulation of q-RRT with parallel quantum computers, and a database building strategy based on simulated annealing. The Parallel Quantum RRT algorithm uses parallel quantum computers in a manager-worker formulation to provide simultaneous measurements of a shared database, allowing more time-efficient tree construction with a higher exploration speed. We also provide key probability results for parallel quantum computers searching the same database in order to predict parallel architecture efficiency. Quantum Database Annealing uses a temperature construct to guide database construction, providing trees that initially spread more

quickly compared to those created with standard database construction, followed by back-fill behavior at lower temperatures. To support these claims, we provide analysis in the form of efficiency and run-time results, heatmaps for speed-of-exploration results, narrow corridor environment results, and database construction comparisons. Future work includes testing combinations of Pq-RRT with QDA as a construction strategy, and expanding on methods of database construction to allow parallel architectures to explore environments faster than Pq-RRT with QDA. Future work also includes creating path planning algorithms that rely on alternate quantum algorithms to QAA.

REFERENCES

[1] P. Lathrop, B. Boardman, and S. Martínez, "Quantum search approaches to sampling-based motion planning," *IEEE Access*, vol. 11, pp. 89506–89519, 2023.
[2] F. Tacchino, A. Chiesa, S. Carretta, and D. Gerace, "Quantum computers as universal quantum simulators: State-of-the-Art and perspectives," *Adv. Quantum Technol.*, vol. 3, no. 3, Mar. 2020, Art. no. 1900052.
[3] S. Pirandola, U. Andersen, L. Banchi, M. Berta, D. Bunandar, R. Colbeck, D. Englund, T. Gehring, C. Lupo, and C. Ottaviani, "Advances in quantum cryptography," *Adv. Opt. Photon.*, vol. 12, no. 4, pp. 1012–1236, 2020.
[4] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.
[5] M. Cerezo, A. Arrasmith, R. Babbush, S. Benjamin, S. Endo, K. Fujii, J. McClean, K. Mitarai, X. Yuan, and L. Cincio, "Variational quantum algorithms," *Nat. Rev. Phys.*, vol. 3, pp. 625–644, Aug. 2021.
[6] D. Abrams and C. Williams, "Fast quantum algorithms for numerical integrals and stochastic processes," 1999, *arXiv:quant-ph/9908083*.
[7] A. Montanaro, "Quantum speedup of Monte Carlo methods," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 471, no. 2181, Sep. 2015, Art. no. 20150301.
[8] R. Portugal, *Quantum Walks and Search Algorithms*, vol. 19. New York, NY, USA: Springer, 2013.
[9] S. Aaronson and A. Ambainis, "Quantum search of spatial regions," 2003, *arXiv:quant-ph/0303041*.
[10] F. Magniez, A. Nayak, J. Roland, and M. Santha, "Search via quantum walk," *SIAM J. Comput.*, vol. 40, no. 1, pp. 142–164, Jan. 2011.
[11] E. Sánchez-Burillo, J. Duch, J. Gómez-Gardeñes, and D. Zueco, "Quantum navigation and ranking in complex networks," *Sci. Rep.*, vol. 2, no. 1, pp. 1–8, Aug. 2012.
[12] C. Petschnigg, M. Brandstötter, H. Pichler, M. Hofbaur, and B. Dieber, "Quantum computation in robotic science and applications," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 803–810.
[13] K. Rajagopal, Q. Zhang, S. Balakrishnan, P. Fakhari, and J. Busemeyer, "Quantum amplitude amplification for reinforcement learning," in *Springer Handbook of Reinforcement Learning and Control*. New York, NY, USA: Springer, 2021, pp. 819–833.
[14] D. Daoyi, C. Chunlin, and L. Hanxiong, "Reinforcement strategy using quantum amplitude amplification for robot learning," in *Proc. Chin. Control Conf.*, Jul. 2006, pp. 571–575.
[15] D. Dong, C. Chen, H. Li, and T. Tarn, "Quantum reinforcement learning," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 38, no. 5, pp. 1207–1220, Oct. 2008.
[16] L. Ming, "An adaptive quantum evolutionary algorithm and its application to path planning," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2015, pp. 2067–2071.
[17] N. B. Dehaghani, F. L. Pereira, and A. P. Aguiar, "Quantum control modelling, methods, and applications," *Extensive Rev.*, vol. 2, no. 1, pp. 75–126, Nov. 2022.
[18] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.
[19] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 1, 1996, pp. 113–120.
[20] J. Barraquand, L. Kavraki, J. Latombe, T. Li, R. Motwani, and P. Raghavan, "A random sampling scheme for path planning," in *Proc. Robot. Res., 7th Int. Symp.* New York, NY, USA: Springer, 1996, pp. 249–264.
[21] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.
[22] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, Jun. 2011.
[23] M. Strandberg, "Augmenting RRT-planners with local trees," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, pp. 3258–3262.
[24] J. Bialkowski, S. Karaman, and E. Frazzoli, "Massively parallelizing the RRT and the RRT*," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 3513–3518.
[25] J. Ichnowski and R. Alterovitz, "Parallel sampling-based motion planning with superlinear speedup," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 1206–1212.
[26] D. Devaurs, T. Siméon, and J. Cortés, "Parallelizing RRT on large-scale distributed-memory architectures," *IEEE Trans. Robot.*, vol. 29, no. 2, pp. 571–579, Apr. 2013.
[27] R. Parekh, A. Ricciardi, A. Darwish, and S. DiAdamo, "Quantum algorithms and simulation for parallel and distributed quantum computing," in *Proc. IEEE/ACM 2nd Int. Workshop Quantum Comput. Softw. (QCS)*, Nov. 2021, pp. 9–19.
[28] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
[29] R. Eglese, "Simulated annealing: A tool for operational research," *Eur. J. Oper. Res.*, vol. 46, no. 3, pp. 271–281, 1990.
[30] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*. Dordrecht, The Netherlands: D. Reidel, 1987.
[31] A. Vázquez-Otero, J. Faigl, and A. Munuzuri, "Path planning based on reaction-diffusion process," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 896–901.
[32] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian optimization for motion planning," *Int. J. Robot. Res.*, vol. 32, nos. 9–10, pp. 1164–1193, Aug. 2013.
[33] S. M. Persson and I. Sharf, "Sampling-based A* algorithm for robot path-planning," *Int. J. Robot. Res.*, vol. 33, no. 13, pp. 1683–1708, Nov. 2014.
[34] L. Jaillet, J. Cortés, and T. Siméon, "Transition-based RRT for path planning in continuous cost spaces," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2008, pp. 2145–2150.
[35] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *IEEE Trans. Robot.*, vol. 26, no. 4, pp. 635–646, Aug. 2010.
[36] P. Lathrop, "Motion planning algorithms for safety and quantum computing efficiency," Ph.D. dissertation, Dept. Mech. Aerosp. Eng., Univ. California, San Diego, CA, USA, 2023.
[37] J. Preskill, "Lecture notes for physics 229: Quantum information and computation," *California Inst. Technol.*, vol. 16, no. 1, pp. 1–8, 1998.
[38] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, "Quantum amplitude amplification and estimation," *Contemp. Math.*, vol. 305, pp. 53–74, Oct. 2002.
[39] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, 2000, pp. 995–1001.
[40] C. Fox, "Quantum computing functions (QCF) for MATLAB," Robotics Res. Group, Oxford Univ., Oxford, U.K., Tech. Rep., 2003.
[41] P. Neal, "The generalised coupon collector problem," *J. Appl. Probab.*, vol. 45, no. 3, pp. 621–629, Sep. 2008.
[42] P. Berenbrink and T. Sauerwald, "The weighted coupon collector's problem and applications," in *Proc. Int. Conf. Comput. Combinatorics*. New York, NY, USA: Springer, 2009, pp. 449–458.

PAUL LATHROP (Student Member, IEEE) received the B.S. degree in aerospace engineering from the University of Maryland, College Park, MD, USA, in 2019, and the Ph.D. degree in aerospace engineering from the University of California at San Diego, in 2023. His research interests include safety and uncertainty in robotic motion planning algorithms and quantum computing and its intersection with motion planning.

BETH BOARDMAN received the B.S. and M.S. degrees in aeronautics and astronautics from the University of Washington, Seattle, WA, USA, in 2010 and 2012, respectively, and the Ph.D. degree in aerospace engineering from the University of California at San Diego, USA, in 2017. She has been a Research and Development Engineer with Los Alamos National Laboratory, since 2018. She is currently the Team Leader of the Cyber-Physical and Robot Innovations Team, Automation, Robotics, and Control Group. She is also the LANL Robotics and Automation Summer School Program Leader. Her research interests include robotics and automation.

SONIA MARTÍNEZ (Fellow, IEEE) received the Ph.D. degree in engineering mathematics from Universidad Carlos III de Madrid, Spain, in May 2002. She was a Visiting Assistant Professor of applied mathematics with the Technical University of Catalonia, Spain, from 2002 to 2003, and a Postdoctoral Fulbright Fellow with the Coordinated Science Laboratory, University of Illinois Urbana–Champaign, from 2003 to 2004, and the Center for Control, Dynamical Systems and Computation, University of California at Santa Barbara, from 2004 to 2005. She is currently a Professor of mechanical and aerospace engineering with the University of California at San Diego, San Diego, CA, USA. She is the coauthor (together with F. Bullo and J. Cortés) of *Distributed Control of Robotic Networks* (Princeton University Press, 2009). She is the coauthor (together with M. Zhu) of *Distributed Optimization-Based Control of Multi-Agent Networks in Complex Environments* (Springer, 2015). Her research interests include the control of networked systems, multi-agent systems, nonlinear control theory, and planning algorithms in robotics. She is the Editor-in-Chief of the recently launched CSS IEEE Open Journal of Control Systems.