NBER WORKING PAPER SERIES

DYNAMIC PROGRAMMING IN ECONOMICS ON A QUANTUM ANNEALER

Jesús Fernández-Villaverde
Isaiah J. Hull

Working Paper 31326
http://www.nber.org/papers/w31326

NATIONAL BUREAU OF ECONOMIC RESEARCH
1050 Massachusetts Avenue
Cambridge, MA 02138
June 2023, revised April 2025

This paper previously circulated as 'Dynamic Programming on a Quantum Annealer: Solving the RBC Model.' We thank participants at numerous seminars and conferences for helpful comments. During part of the project, Isaiah Hull was affiliated with and held a financial interest in CogniFrame, Inc., a partner of D-Wave Systems, the producer of the quantum annealers used in this paper. The authors received no support or computing time from D-Wave through this partnership. All access to D-Wave's annealers was obtained via a standard developer agreement. The views expressed herein are those of the authors and do not necessarily reflect the views of the National Bureau of Economic Research.

Dynamic Programming in Economics on a Quantum Annealer
Jesús Fernández-Villaverde and Isaiah J. Hull
NBER Working Paper No. 31326
June 2023, revised April 2025
JEL No. C63, C78, E37

ABSTRACT

We introduce novel algorithms for solving dynamic programming problems in economics on a quantum annealer, a specialized quantum computer used for combinatorial optimization. Quantum annealers begin in a superposition of all states and generate candidate global solutions in milliseconds, regardless of problem size. In contrast to existing methods, those developed in this paper 1) recover value and policy functions, 2) avoid fundamental scaling bottlenecks, and 3) are already implementable for small (but non-trivial) problems on current hardware. We demonstrate the method by solving the real business cycle model on a quantum annealer.

Jesús Fernández-Villaverde
Department of Economics
University of Pennsylvania
The Ronald O. Perelman Center
for Political Science and Economics
133 South 36th Street Suite 150
Philadelphia, PA 19104
and CEPR
and also NBER
jesusfv@econ.upenn.edu

Isaiah J. Hull
Department of Finance, BI Norwegian Business School
Nydalsveien 37, 0484 Oslo, Norway
hull.isaiah@gmail.com

1 Introduction

Our point of departure is a well-known challenge: solving dynamic programming problems (DPPs) on classical (non-quantum) hardware is severely constrained by the "curse of dimensionality" (Bellman, 1957, p. IX). As the number of state variables and the granularity of the grid increase, the time and memory requirements of classical algorithms grow exponentially. Thus, the set of tractable dynamic optimization models in economics remains narrow, limited to small problems or those with special structural properties, such as known functional forms.

We explore quantum computing—an emerging but still maturing technology—as a path to solving larger DPPs. Specifically, we introduce a new class of algorithms for solving such problems on a quantum annealer (QA), a specialized quantum device that can be used to perform combinatorial optimization. To the best of our knowledge, this is the *first* class of quantum algorithms designed to solve DPPs commonly encountered in economics, which typically require the recovery of a policy or value function.

The algorithms we introduce differ fundamentally in nature and structure from classical dynamic programming methods. Although QAs do not permit theoretical proofs of scaling advantage, we demonstrate their competitive performance on DPPs through a series of experiments conducted on actual hardware.[1] Our contribution, therefore, is primarily the development of quantum algorithms that enable the solution of economic DPPs on quantum hardware.

The special purpose quantum computer we use in this paper embeds the parameters of a combinatorial optimization problem in a quantum system that evolves to its lowest energy configuration (Venegas-Andraca et al., 2018). This is equivalent to finding the set of variable values that globally minimize a loss function (Farhi et al., 2000). Since the duration of this physical process does not depend on the size of the state space, the execution time is effectively constant in the problem size, but is subject to a constraint on the maximum problem size. Additionally, unlike classical methods, QAs operate by initializing the system in a quantum superposition of all possible states, rather than a single state, and return candidate global solutions in milliseconds.

We implement these algorithms on quantum hardware, rather than using classical simulations, and show that they are already competitive with state-of-the-art classical methods for a small but non-trivial dynamic equilibrium problem: the real business cycle (RBC) model. Additionally, we introduce the concepts and techniques in quantum computing needed to apply this approach more broadly to other problems in economics and finance.

[1] On universal quantum computers (UQCs), it is possible to execute algorithms with provable scaling properties, which may differ from the equivalent classical (non-quantum) algorithm. For certain problems, the required computational resources grow exponentially on a classical computer, but only polynomially on a quantum one. The class of problems solvable by a quantum computer with bounded error in polynomial time is referred to as "bounded-error quantum polynomial time," or BQP.

Our argument above is built around three core components: DPPs, QAs, and the RBC model. We justify each in turn.

We begin with DPPs, for two reasons. First, improving the efficiency of solving even medium-scale DPPs would generate substantial gains across many areas of economics, including macroeconomics, labor, and industrial organization. Second, despite their importance, progress on DPPs within the quantum computing literature has been limited. Most existing results are either not applicable to economic DPPs or are not implementable on current hardware.[2]

Next, we focus on quantum annealers (QAs), as they are presently the only quantum computing devices with the potential to solve economic DPPs on actual quantum hardware. As discussed in Sections 2 and 3, QAs are special-purpose quantum computers that can be used to solve combinatorial optimization problems. In contrast, universal quantum computers (UQCs) are theoretically capable of executing arbitrary quantum algorithms, but are currently more limited in their practical capabilities.

QAs do, however, present significant challenges for our purposes. They do not natively allow for iteration over time or across multiple objective functions and, similar to UQCs, are subject to the quantum-to-classical bottleneck, which severely limits how much classical information can be read out from a computation.[3] A central contribution of this paper is to reformulate economic DPPs in a way that can be executed on a QA despite these constraints.

The approach we introduce addresses these limitations directly. In particular, our algorithms recover the key economic objects of interest: policy and value functions. This stands in contrast to much of the existing quantum computing literature, which typically focuses on recovering decision paths or other low-dimensional objects of interest in other fields.[4]

Finally, we demonstrate our approach using the RBC model. This model sits at the boundary of what current quantum hardware can handle. Indeed, hardware constraints make solving the RBC model infeasible on a quantum device without the algorithms and advanced annealing strategies introduced in this paper. While our algorithms perform competitively with state-of-

[2]The theoretical quadratic speed-up proposed by Ambainis et al. (2019) is not yet implementable on existing devices. It combines Grover's unstructured quantum search (Grover, 1996) with a partial dynamic programming table to accelerate solving subset selection problems on universal quantum computers.

[3]A QA with N qubits begins in a quantum superposition of size 2^N, but ultimately collapses into a classical state containing only N bits of information. While this bottleneck is not unique to QAs—it affects all quantum computers—it poses a particular challenge for DPPs in economics, which require encoding richer solution structures into a small number of classical bits.

[4]The traveling salesperson problem (TSP) is one of the most studied DPPs in quantum computing. In the TSP, we are given a list of cities, the distances between each pair, and a starting point. The goal is to find the shortest route that allows a salesperson to visit each city exactly once and return to the starting city. Finding this route is an NP-hard problem, which implies that it is at least as hard to solve as any other problem in NP. While the computational resources required to solve the TSP grow exponentially with the number of cities, the solution itself —a sequence of cities— grows only linearly. Experimental work on the TSP (e.g., Heim et al., 2017; Kieu, 2019; Warren, 2019) is not directly applicable to economic DPPs, which typically require recovering decision rules or value functions.

the-art classical methods, reducing execution time is not our primary objective. Rather, solving the RBC model demonstrates how to formulate DPPs for a QA and highlights the feasibility and efficiency of our approach using existing hardware.

To benchmark our algorithms, we compare execution times and error sizes to: i) the classical value function iteration (VFI) solution in Aruoba and Fernández-Villaverde (2015) (hereafter, AFV); and ii) parametric policy iteration (PPI) (Benítez-Silva et al., 2000). We choose VFI since it is the standard reference in DPPs. Readers can compare their preferred algorithm (e.g., a parallelized projection method) to VFI and, by extension, to our QA algorithm. We select PPI for its efficiency in solving the RBC model.

Despite current quantum hardware limitations, our algorithms yield solutions accurate enough for standard applications and with competitive execution times. As emphasized above, the execution time of our algorithms does not grow with problem size; however, the maximum problem size is constrained by the hardware. Future hardware improvements will allow us to solve larger problems in *exactly the same time*. This contrasts with classical methods, whose execution time grows exponentially. These results are evidence of the *potential* of quantum hardware for economists, both in designing and implementing new algorithms on QAs.

In summary, the solution algorithms we introduce are i) free of scaling issues that would prevent their future use on larger problems; and ii) implementable for small problems on current hardware. This addresses two common pitfalls in the design of quantum algorithms. First, many theoretical advances demonstrate speed-ups using algorithms that are not feasible to implement on existing hardware and may not be implementable in the foreseeable future. Second, many proof-of-principle hardware demonstrations—which solve toy problems to show feasibility—have known scaling limitations that make them unsuitable for larger problems.

While our focus is DPPs in economics, our algorithms apply more broadly. They can solve both native combinatorial optimization problems and those that can be reformulated as such.[5] They also offer a framework for tackling iterative and multi-objective optimization problems on QAs, using novel applications of advanced annealing techniques to enable iteration within and across anneals. Specifically, we demonstrate how to combine reverse and inhomogeneous annealing to alternate over components of a multi-objective loss function.[6] Furthermore, the algorithms we introduce are efficient, achieving run times near the theoretical minimum of what is possible for the annealer used in our application.

The rest of the paper is organized as follows. Section 2 introduces universal quantum computing and its applications in economics and finance. Section 3 covers quantum annealing. Section 4 presents our testbed: the benchmark RBC model. Section 5 explains how to translate the model into a form solvable on a QA. Section 6 introduces hybrid and quantum algorithms

[5]The DPP we solve is not natively combinatorial and requires a non-trivial reformulation to run on a QA.
[6]QAs do not natively support iteration, which typically requires hybridization with classical components.

for solving the model and compares them to the benchmarks in AFV. Section 7 concludes. An Online Appendix provides further details.

2 Universal Quantum Computing

There are two broad categories of quantum computers: universal quantum computers (UQCs) and special-purpose quantum computers. UQCs can, in principle, run arbitrary algorithms. In contrast, special-purpose quantum computers typically solve a specific class of problems. QAs, for example, solve combinatorial optimization problems by finding the lowest energy state of a quantum system.[7] This section briefly reviews UQCs and their applications in economics and finance. The rest of the paper focuses on quantum annealing.

The gate-and-circuit model. UQCs typically use the "gate-and-circuit" model, where quantum operations called gates are applied to quantum bits (qubits) to perform computations. Qubits encode information and quantum gates modify it —just as classical computers encode information in bits and use logic gates to modify it. However, quantum gates must adhere to the laws of quantum physics, as they operate on quantum systems. A quantum circuit is a sequence of such gates designed to carry out a specific task, like running an algorithm.

Computational complexity. Since UQCs offer theoretical reductions in computational complexity, they may eventually solve problems that are infeasible for any classical computer, including supercomputers or GPU arrays. These gains arise from exploiting quantum resources that are unavailable to classical machines to achieve better scaling properties than the best available classical algorithm for the same task.

Integer factorization was one of the first tasks for which a quantum scaling advantage was identified. On classical computers, the number of non-parallelizable steps needed to factor large integers grows nearly exponentially with input size. However, on a UQC, the number of steps scales only polynomially using the quantum algorithm introduced in Shor (1997).[8] Thus, a UQC could eventually make factoring large integers practical—something that remains out of reach using the best available classical algorithms and hardware.

Applications in economics and finance. Analyzing the computational complexity of quantum algorithms for UQCs enables us to determine whether quantum computing offers scaling

[7]For instance, D-Wave's quantum annealers can be used to solve binary quadratic models (BQMs), whereas QuEra's neutral atom arrays solve certain graph-based problems, such as maximum independent set (MIS) problems. Annealer-based models of universal quantum computing have been theorized (Ozfidan et al., 2020; Imoto et al., 2024), but are not the focus of annealing applications in this paper.

[8]The algorithm proposed by Shor (1997) is not yet implementable on existing quantum hardware. Moreover, a classical algorithm may still be discovered that factors integers in polynomial time.

advantages over classical approaches, even while the hardware remains underdeveloped. This is why much of the current research on quantum computing, including in economics and finance, focuses on theory, where current hardware limitations are less relevant.

In finance, there has been progress in exploring theoretical quantum speed-ups. For instance, the quantum amplitude estimation (QAE) algorithm provides a theoretical quadratic speed-up for Monte Carlo simulations. Woerner and Egger (2019) use QAE for credit risk analysis. Egger et al. (2021) and Ghysels et al. (2023) apply it to computing distribution quantiles, which yield value-at-risk and expected shortfall measures. QAE has also been used in pricing exotic options under Black-Scholes-type models (Stamatopoulos et al., 2020) and in models with stochastic volatility (Kaneko et al., 2022; Ghysels et al., 2023; Carrera Vazquez and Woerner, 2021).

Another theoretical speed-up, proposed by Harrow et al. (2009), offers an exponential reduction in the time complexity of solving linear systems. Ghysels and Morgan (2024) apply it to non-linear asset pricing models and provide an implementation that runs on current hardware. Beyond the speed-up, their work links the quantum operator-theoretic framework to decision theory, ambiguity aversion, model and parameter uncertainty, and risk. For overviews of algorithms in computational economics, econometrics, and finance that have provable quantum speed-ups, see Hull et al. (2020) and Herman et al. (2022).

DPPs. There have also been advances in solving DPPs using quantum computers. Ambainis et al. (2019) and Glos et al. (2021) prove theoretical quantum speed-ups for several problem classes, including the traveling salesperson and vertex ordering problems. These speed-ups are quadratic and thus still limited by the underlying exponential time complexity, but may extend the size of solvable instances. However, they remain difficult to implement on current hardware and are not directly suited to the types of DPPs common in economics and finance.

3 Quantum Annealing

In contrast to UQCs, QAs cannot execute arbitrary algorithms or offer provable reductions in computational complexity. Instead, they are limited to applications involving combinatorial optimization, sampling, and simulating physical systems. This narrower scope has enabled significantly faster hardware development. At the time of this paper's writing, QAs had roughly 50 times more qubits than publicly available UQCs.[9]

[9]The largest publicly accessible UQCs, produced by IBM, include the *Eagle* processor (127 qubits), and the exploratory *Osprey* (433 qubits) and *Condor* (1121 qubits) devices. In contrast, D-Wave's *Advantage* annealer, used in this paper, features 5616 qubits. This substantial size difference led us to use a QA instead of a UQC and the quantum approximate optimization algorithm (QAOA), which can also solve combinatorial optimization problems (Farhi et al., 2014).

In the remainder of this section, we introduce quantum annealing, outline the types of problems it can address, explain its limitations, and highlight potential applications in economics. Appendix A.1 discusses the computational complexity of the optimization problem that annealers solve, along with existing evaluations of their performance. We begin with a brief overview of combinatorial optimization.

3.1 Combinatorial Optimization

Quantum annealing is a heuristic method for solving combinatorial optimization problems that may offer a significant computational advantage over classical methods in specific cases (Zintchenko et al., 2015). For our purposes, we adopt a slightly modified definition of a combinatorial optimization problem from Venegas-Andraca et al. (2018).

Definition 1 (Combinatorial Optimization Problem)**.** *Let E be a finite set with cardinality $|E| = n$, P_E its power set (so $|P_E| = 2^n$), and $C : P_E \to \mathbb{R}$ a loss function. The goal is to find $\mathcal{P} \in P_E$ such that $C(\mathcal{P}) = \min_{\mathcal{P}_i \in P_E}\{C(\mathcal{P}_i)\}$.*

In game theory and mechanism design, problems involving the allocation of discrete objects often take a natural combinatorial form suited to quantum annealing. Financial network problems also fall under this framework and have been approached using QAs in proof-of-principle studies by Orús et al. (2019a,b).

Our focus is on economic problems that do not naturally match the definition above but are often reformulated as combinatorial problems to enable grid search or value function iteration. Consider first a firm minimizing a cost function C by choosing inputs L and K, subject to the production constraint $y^* = Y(L, K)$. If this problem lacks a closed-form solution, we can solve it by searching over the Cartesian product of two discrete grids: $K \in \{K_0, K_1, ..., K_{\bar{K}-1}\}$ and $L \in \{L_0, L_1, ..., L_{\bar{L}-1}\}$. The solution is a pair of functions, $K^*(w, r, y)$ and $L^*(w, r, y)$, that yield optimal input choices for capital and labor given factor prices (w, r) and target output y.

As a second example, consider an infinitely lived household that maximizes utility from consumption, subject to a budget constraint. This problem is typically formulated as a DPP and solved using value function iteration. The result is a discrete look-up table that approximates the optimal value function or policy rule for capital and consumption choices.

While both examples can, in principle, be solved on a QA once reformulated as combinatorial problems, the limitations of quantum computers make this challenging. The main obstacle—explained in Subsection 3.4—is the quantum-to-classical bottleneck, which limits how much information can be extracted from a quantum device, including a QA. These problems also differ in computational complexity from standard combinatorial optimization tasks, as discussed in Appendix A.1.

3.2 Adiabatic Quantum Computing

Quantum annealing can be viewed as a *heuristic* implementation of adiabatic quantum computing (Venegas-Andraca et al., 2018; Shin et al., 2014), whose usefulness as an optimization algorithm is motivated by the quantum adiabatic theorem (Messiah, 1958; Ambainis and Regev, 2004). To clarify these ideas, we outline the idealized framework of adiabatic quantum computing, which may be interpreted as a best-case scenario for quantum annealing.

Ambainis and Regev (2004) offer the following informal version of the *quantum adiabatic theorem*: *"... if we take a quantum system whose Hamiltonian slowly changes from $\mathcal{H}_1$ to $\mathcal{H}_2$, then, under certain conditions on $\mathcal{H}_1$ and $\mathcal{H}_2$, the ground (lowest energy) state of $\mathcal{H}_1$ gets transformed to the ground state of $\mathcal{H}_2$."* A Hamiltonian describes the total energy in a quantum system, where the operator's eigenvalues represent energy levels of the corresponding eigenvectors.

Farhi et al. (2000) demonstrated that this principle can be used to build an alternative model of quantum computation based on adiabatic evolution for global minimization. In this setup, a quantum state evolves between two Hamiltonians: (1) a simple, analytically solvable Hamiltonian $\mathcal{H}_0$, and (2) a problem Hamiltonian $\mathcal{H}_p$. The system is initialized in the ground state of $\mathcal{H}_0$ and gradually transitions to $\mathcal{H}_p$ over a time interval T, following the path:

$$\mathcal{H}(t) = \left(1 - \frac{t}{T}\right)\mathcal{H}_0 + \frac{t}{T}\mathcal{H}_p.$$

The initial Hamiltonian $\mathcal{H}_0$ is specified to be trivial with a known ground state, while $\mathcal{H}_p$ encodes the optimization problem of interest, where the unknown ground state coincides with the global minimum. Thus, both $\mathcal{H}_0$ and $\mathcal{H}_p$ can be seen as loss functions, with ground states corresponding to their respective minimizers.

Returning to the quantum adiabatic theorem: evolving from $\mathcal{H}_0$ to $\mathcal{H}_p$ slowly enough ensures the system stays in its ground state throughout. In principle, this provides a method for finding the global minimum of a combinatorial optimization problem.

Adiabatic quantum computing also offers a "speed limit" for the transition from $\mathcal{H}_0$ to $\mathcal{H}_p$ while maintaining the system in its ground state. Unfortunately, for many problems, this takes the form of $T = \mathcal{O}[\exp(\gamma N^\nu)]$, where γ and ν are positive constants and N denotes the problem size (Bapst et al., 2013; Lucas, 2014). Thus, adiabatic quantum computing may still require exponential time to solve combinatorial optimization problems. Yet, if γ and ν are much smaller than their classical counterparts, QAs might still be able to solve larger problem instances. We revisit this point in Appendix A.1.

3.3 Problem Encoding

Quantum annealing evolves a trivial Hamiltonian, $\mathcal{H}_0$, initialized in its ground state, into a problem Hamiltonian, $\mathcal{H}_p$, which encodes the objective function. If the transition is adiabatic (i.e., slow enough), the system remains in the ground state throughout. Measuring the system at the end of the process yields the global minimum.

QAs do not require a specific choice of $\mathcal{H}_0$, as any trivial Hamiltonian with a known ground state will suffice.[10] However, specifying $\mathcal{H}_p$ is essential—and often challenging—even for simple problems. This task, known as "problem encoding," maps a minimization problem onto a quantum system. We now turn to how such encodings are constructed, beginning with which problems can feasibly be encoded.

3.3.1 Conversion to a binary quadratic model

The hardware and topology of a QA require encoding optimization problems as binary quadratic models (BQMs). This can be done using either the Ising model or the quadratic unconstrained binary optimization (QUBO) model. We introduce the Ising model first, as it offers transparency and intuitive appeal. However, the QUBO model will be our preferred framework, as it abstracts from physical systems and requires no background in physics.[11]

The Ising model. The Ising model describes a system of atomic spins. For our purposes, it is enough to understand that each spin is a physical system with two possible states, denoted +1 and -1. We can interpret a spin as representing a terminal qubit state in a QA.

We denote a vector of N spins as $s = \{s_0, s_1, ..., s_{N-1}\}$. Each s_i has a bias h_i, and each pair (s_i, s_j) has a coupling $J_{i,j}$, with $h_i, J_{i,j} \in \mathbb{R}$. These define the Ising model:

$$\mathcal{H}(s) = \sum_{i=0}^{N-1} h_i s_i + \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} J_{i,j} s_i s_j.$$

Here, $\mathcal{H}$ represents a loss function as the system's total energy. Higher $\mathcal{H}$ means higher loss. The biases and couplings encode the optimization problem. The annealing process seeks a spin configuration that minimizes energy.

For example, let $N = 2$, $h = \{0.5, -0.3\}$, and $J = \{-0.8\}$. Table 1 lists all terminal s vectors and their total energy. Combination 1, with $s = \{-1, -1\}$, is the global minimum, since $\mathcal{H}(-1, -1)$ is the lowest. While trivial for $N = 2$, enumerating all combinations for

[10] In practice, an equal superposition of all states is common. "Reverse" annealing can also begin from a classical state, enabling local refinement of solutions.

[11] Throughout this paper, *quadratic* refers to the number of pairwise interactions between binary variables, not to quadratic approximations in perturbation theory.

	s_0	s_1	$\mathcal{H}(s)$
1	-1	-1	-1.0
2	-1	1	0.0
3	1	-1	1.6
4	1	1	-0.6

Table 1: Spin configurations and energy levels.

arbitrary N has classical time complexity $\mathcal{O}(2^N)$.

BQMs, including the Ising model, have a natural graphical representation. Consider the case for a graph of size $N = 4$ with non-zero couplings between the pairs (s_0, s_1), (s_0, s_3), (s_1, s_2), and (s_1, s_3). Abstracting from biases and coupling magnitudes, we can represent this model with a graph, G, specified by vertices and edges, (V, E). Figure 1 visualizes the graph, where $V = \{s_0, s_1, s_2, s_3\}$ and $E = \{(s_0, s_1),(s_0, s_3),(s_1, s_2),(s_1, s_3)\}$.

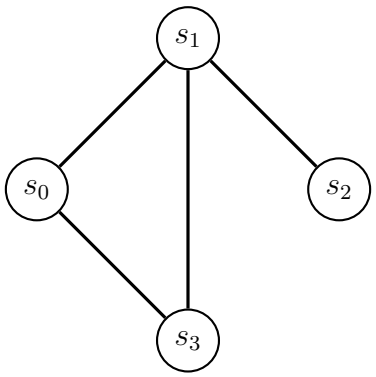


Figure 1: A graphical representation of a binary quadratic model for the $N = 4$ case and with couplings between the pairs: $\{(s_0, s_1),(s_0, s_3), (s_1, s_2),(s_1, s_3)\}$.

As we discuss in Section 5, the BQM derived from our problem must be mapped onto a graph that can be embedded in a QA. While this embedding is typically generated as a classical pre-processing step —and is not part of the quantum solution— the problem's difficulty and the quality of the embedding depend on the structure of the underlying graph (Lucas, 2014). Section 5 examines whether the BQM's graph is compatible with the topology of a given QA.

Earlier, we introduced the classical Ising model to build intuition. We described the Hamiltonian as an operator that expresses the system's energy as a function of its state and the parameters of the system, which can be configured to encode an optimization problem. A quantum version of the Ising model is presented in Appendix A.

The QUBO model. The QUBO model is a BQM, similar to the Ising model, but it uses 0 and 1 states rather than the +1 and -1 states of the Ising model. A QUBO formulation allows

for both equality and inequality constraints and does not rely on a quadratic approximation.[12]

The standard form of a QUBO problem is:

$$\mathcal{H}_p(x) = \sum_{i=0}^{N-1} Q_{i,i} x_i + \sum_{j<i}^{N} Q_{i,j} x_i x_j.$$

Notice that $x = \{x_0, x_1, \ldots, x_{N-1}\}$ is a vector of binary variables, each taking values in $\{0, 1\}$. The problem structure is embedded in Q, an upper-triangular matrix. The objective function can be rewritten as $\mathcal{H}_p(x) = x^T Q x$.

Model conversion. Converting the constrained optimization problem implied by an economic model into a BQM is not straightforward. As discussed next, we take an intermediate step: mapping the objective function to a pseudo-Boolean function (PBF). We then show how to decompose the PBF into a QUBO and expand the objective function to incorporate constraints.

3.3.2 Pseudo-Boolean functions

NP-hard problems must typically be converted to a PBF before they can be reduced to a QUBO (Venegas-Andraca et al., 2018). This includes most optimization problems in economics and finance that are suitable for quantum annealing. A PBF is a function $f : \mathcal{B}^N \to \mathbb{R}$, where $\mathcal{B}$ is the Boolean domain.

Boros and Hammer (2002) show that every PBF admits a unique representation as a multilinear polynomial:

$$f(x) = \sum_{\mathcal{M} \subseteq \{1,\ldots,N\}} \alpha_{\mathcal{M}} \prod_{i \in \mathcal{M}} x_i,$$

which can be reduced to a QUBO in polynomial time and with a polynomial bound on problem size.

Thus, once an optimization problem is reformulated as a PBF, it can be recast as a QUBO and potentially solved on a quantum annealer. This reformulation is often nontrivial, as most complex economic problems do not naturally map to QUBOs, but many can be cast as PBFs.

Partial example of PBF mapping. For illustration, consider an optimization problem where we choose next-period capital, k', given current capital, k, and productivity, z. To solve a discrete approximation of the problem, we define a grid for z and a shared grid for k and k', mapping indices to variable values. A scalar-valued loss function then takes these indices, (i, j, m), as inputs.

[12] QUBO models allow constraints, but they must be incorporated into the objective function. As we will see, this does not prevent us from including higher-order terms, which can be incorporated after *quadratization*.

Recasting this loss function as a QUBO presents two challenges. First, QUBOs are restricted to pairwise interactions between binary variables, yet our function involves three indices. Second, QUBOs use binary rather than integer-valued variables.

A simple, though inefficient, solution to the second problem maps each node in the grids for k, z, and k' to a distinct binary variable. If k is at node j on its grid, then we set $x_{k_j} = 1$ and $x_{k_l} = 0$ for all $l \neq j$. In this way, the original index triplet (i, j, m) can be represented by $x_{k_i} x_{z_j} x_{k'_m} = 1$ in a PBF. But since this is a cubic term, it cannot appear directly in a QUBO.

One solution is to *quadratize* the term: rewrite it as an equivalent expression involving only quadratic and lower-order terms. We explain how to do this next.

3.3.3 Degree reduction

Once the initial problem has been mapped to a PBF, the next step is to perform degree reduction to eliminate all higher-order terms. A good quadratization method yields a QUBO with the same global minimum as the PBF while minimizing the number of auxiliary variables introduced.

Dattani (2019) and Dattani and Chancellor (2019) evaluate over 30 quadratization methods. In Online Appendix B, we discuss four that introduce either no auxiliary variables or the minimum required. The method introduced below applies universally but at the cost of requiring more auxiliary variables. All selected methods are compatible with D-Wave's *Pegasus* QA topology, which we use in our application.

Reduction by substitution. A general reduction strategy replaces products of two variables in higher-order terms with an auxiliary variable and a quadratic constraint. Consider the PBF:

$$\mathcal{H}_p(x) = 2x_1x_2x_3 + 4x_2x_3x_4 - 5x_2x_3x_5.$$

Since x_2x_3 appears in every term, substituting $x_a = x_2x_3$ reduces all terms to quadratic form. This requires a constraint to enforce the substitution. The new Hamiltonian becomes:

$$\mathcal{H}'_p(x) = 2x_1x_a + 4x_ax_4 - 5x_ax_5 + \gamma \underbrace{(x_2x_3 - 2(x_2 + x_3)x_a + 3x_a)}_{x_a = x_2x_3}.$$

Here, $\gamma > 0$ is a penalty weight chosen by the researcher.[13]

While this approach offers a general-purpose method for degree reduction, it has a significant cost: one auxiliary variable and one penalty term per reduced degree. The penalty parameter is particularly problematic, as QAs are sensitive to the magnitude of the largest model parameter.

[13]The substitution replaces x_2x_3 with x_a in the objective. To enforce $x_2x_3 = x_a$, we add a penalty term. When x_2, x_3, and x_a are binary, the penalty $\gamma(x_2x_3 - 2(x_2 + x_3)x_a + 3x_a)$ discourages deviation from $x_2x_3 = x_a$.

Choosing a large enough γ to enforce the constraint can dilute the relative strength of couplings tied to the original problem.

3.3.4 Adding constraints

The term *unconstrained* in quadratic unconstrained binary optimization refers to the absence of external constraints beyond the QUBO objective. However, constraints can still be incorporated into the QUBO by embedding them in the objective function. For example, consider the objective $f(x) = x_1x_2x_3 + 3x_1x_3 + 2x_2$ with the constraint $x_1 + x_2 = 1$. We can rewrite this as:

$$\mathcal{H}(x) = x_1x_2x_3 + 3x_1x_3 + 2x_2 + \gamma(1 - x_1 - x_2)^2.$$

3.4 Limitations

Quantum computers can execute algorithms that require less time and space than the best classical alternatives for certain problems. However, they also impose unfamiliar constraints. This subsection highlights the most relevant limitations for our application.

Quantum-to-classical bottleneck. QAs exploit quantum superposition, allowing qubits to exist in a linear combination of the 0 and 1 states—unlike classical bits, which are in either 0 or 1. For instance, two classical bits can be in one of four states: 00, 01, 10, or 11. In contrast, two qubits can be in a linear combination of all four states simultaneously. More generally, an N-qubit system can be in a linear combination of all 2^N possible states.

This property allows us to initialize a QUBO problem in a uniform superposition over all 2^N states, rather than in any single one.[14] During annealing, the system evolves from this quantum superposition to a pure or basis state, ideally corresponding to the solution. The QA then performs measurement and reads out this final classical state.

The fact that we cannot directly observe a quantum state—and must instead sample from it via measurement—is known as the *quantum-to-classical bottleneck*. Even if we create a superposition over 2^N states, each annealing step yields only N classical bits. With a state-of-the-art QA featuring 5000 qubits, each run returns at most 0.625kB of classical information. As a result, quantum annealing is best suited to problems with space-efficient solutions.

Classical-to-quantum overhead. While the quantum annealing step takes at least $5\mu s$ (microseconds), this duration does not grow with problem size.[15] However, programming the quantum processing unit (QPU) requires 9ms, and readout adds another $120\mu s$ (Venegas-Andraca

[14] For an equal superposition, each of the 2^N states has an equal probability weight of $\frac{1}{2^N}$ prior to measurement. Upon measurement, the superposition collapses, and the system moves into one of these states randomly.

[15] We use a $20\mu s$ anneal time for forward annealing, as recommended for the hardware in this paper.

et al., 2018). Thus, total run time is $T = T_p + R(T_a + T_r)$, where T_p is programming time, T_a is annealing time, T_r is readout time, and R is the number of repetitions.

The QPU is programmed only once, but the annealing and readout steps are repeated R times. Programming takes roughly 72 times longer than annealing and readout combined. This creates significant overhead in hybrid algorithms that alternate between classical and quantum steps.

Because quantum annealing is probabilistic, each run may yield a different outcome. Hence, we typically perform many repetitions ($R > 1$) and analyze the frequency and energy (loss) levels of different results. This also mitigates the importance of the transfer overhead.[16]

Noise. Unlike classical machines, quantum computers cannot yet perform error correction or store information for extended periods. This is due to i) the impossibility of copying quantum states (Wootters and Zurek, 1982); and ii) the tendency of quantum systems to decohere in nanoseconds or microseconds. In this regard, QAs have an edge over universal quantum computers, as the annealing process is less affected by qubit decoherence (Childs et al., 2001; Albash and Lidar, 2015).

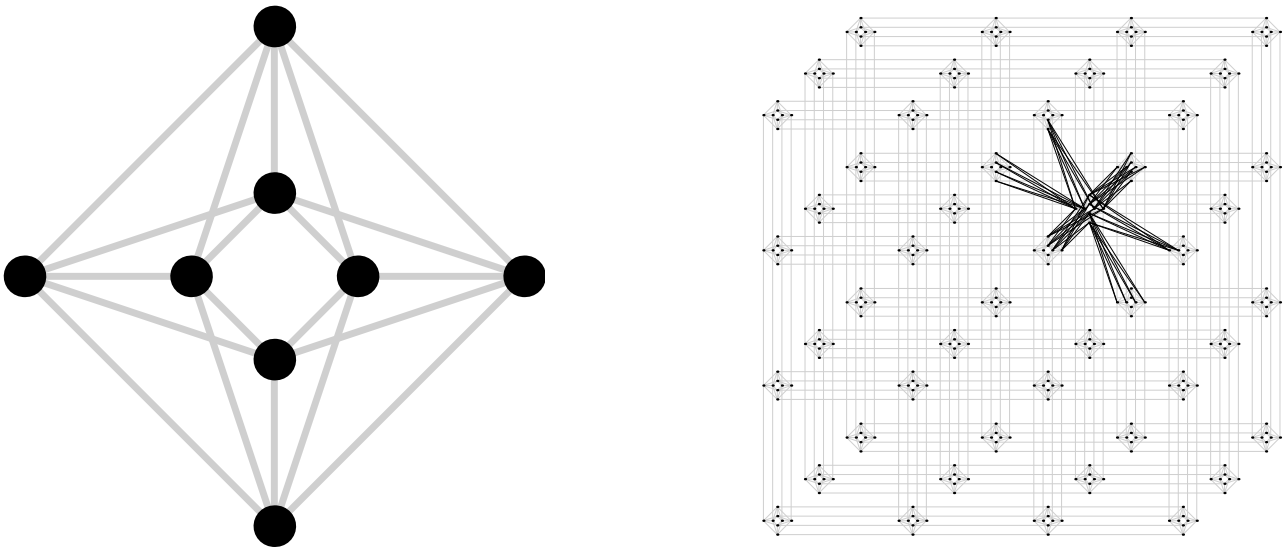
QPU topology. Once the initial problem is reduced to a QUBO, it must be embedded in the QPU's graph. The graph's topology imposes further constraints on the problems that can be solved with a QA. Figure 2 shows a cell and subgraph from the *Pegasus* topology, used in D-Wave's *Advantage* QAs.

The primary topological considerations are the total number of qubits, the number of couplers, and the number of couplers per qubit. Having fewer qubits limits the number of binary variables we can represent in a QUBO and reduces the amount of classical information we can extract. Likewise, having fewer couplers restricts the use of *quantum entanglement* as a computational resource, which is essential for encoding the strength and direction of relationships between variables.

In addition to the total number of couplers, the number of couplers per qubit also matters, as it determines how much *chaining* is needed. Chaining occurs when two qubits in a QUBO share a quadratic term but are not directly coupled. It creates an indirect link by forcing intermediary qubits to match the state of one of the connected qubits. Long chains reduce the number of available qubits to represent the problem and require tuning a *chain strength* hyperparameter.[17]

[16] Standard practice is to select the lowest-energy solution. However, below, we solve a multi-objective problem iteratively, zeroing out one component of the objective across anneals. In those cases, the final energy level may not reflect both components.

[17] In quantum computing, it is common to distinguish between "physical" and "logical" qubits. A physical qubit is an actual quantum bit on the processor. A logical qubit is an abstract unit made of several physical



Intra-Cell Connections

Inter-Cell Connections

Figure 2: Intra-cell and inter-cell connections on subgraphs of the 16×16×3 *Pegasus* topology. The left panel shows connections within an eight-qubit cell. The right panel displays inter-cell connections for a sample cell embedded in a 4×4×3 subgraph. The figure was generated using the method in Dattani et al. (2019).

3.5 Ideal Problems

Given the features of a QA, what makes an ideal problem to demonstrate quantum advantage? The literature identifies three key attributes: 1) a large state space with a space-efficient solution; 2) sharp spikes in the loss function (energy landscape); and 3) many local minima between the starting point and the global minimum.

Large state space and space-efficient solution. A problem solved directly on a QPU without partitioning into subproblems should require no more than N bits to express its solution, where N is the number of qubits. D-Wave's *Advantage* QAs, for example, have over 5000 qubits and can output at most 0.625kB of classical information per anneal. An ideal problem would exploit quantum superposition to search a large state space but yield a solution that fits within a small classical output. A challenging static equilibrium-finding problem, where only a price and quantity need to be reported, is a good example.

Spikes in the loss function. QAs exhibit *quantum tunneling*, where a qubit passes through a barrier in the energy landscape (loss surface), rather than climbing over it, as classical algorithms must (Boixo et al., 2014; Albash et al., 2015b,a). Denchev et al. (2016) suggest that problems with loss functions featuring long, thin barriers (spikes) around local minima are well-suited for

qubits. Logical qubits are used for error correction or to work around device limitations. In QA, chaining creates logical qubits to compensate for limited connectivity.

tunneling. Figure 3 shows the loss function of a problem that may benefit from a tunneling-based speed-up. Classical solvers get stuck in local minima, while QAs may tunnel through the barriers to reach the global minimum.

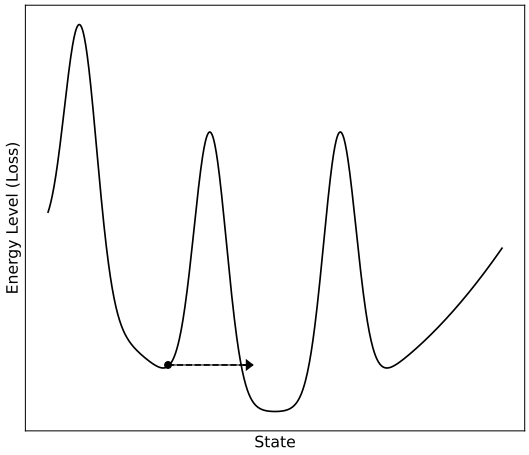


Figure 3: *Quantum tunneling* is depicted by the dashed line passing through the energy landscape.

Many local minima on the path to the global minimum. Quantum annealing also exhibits *co-tunneling*, where multiple entangled qubits tunnel jointly through a barrier to a lower energy level (loss) (Boixo et al., 2016). Without entanglement, this simultaneous tunneling would be impossible. This phenomenon can help navigate complex loss surfaces by bypassing several local minima at once.[18]

4 The Model

To illustrate the potential of QAs in economics, we solve the version of the RBC model used in AFV, a standard benchmark in computational economics. Given the limitations of current quantum hardware, solving this model on a QA is highly challenging. It requires several methodological advances and the novel application of advanced annealing techniques.

A social planner chooses a sequence of consumption c_t and capital k_{t+1} to solve:

$$\max_{\{c_t, k_{t+1}\}} \mathbb{E}_0 \sum_{t=0}^{\infty} (1-\beta)\beta^t \ln(c_t)$$

[18]Ideal cases for co-tunneling involve regions of the loss surface where changing one variable at a time increases loss, but changing all of them together reduces it.

subject to the resource constraint $c_t + k_{t+1} = z_t k_t^\alpha + (1-\delta)k_t$, where δ is the depreciation rate, α is the output elasticity of capital, β is the discount factor, z_t is productivity, and $\mathbb{E}_0$ is the conditional mathematical expectation.

To match the comparison exercise in AFV and obtain a closed-form benchmark for our QA solution, we set $\beta = 0.95$, $\alpha = 0.33$, and $\delta = 1$. With full depreciation, the social planner's optimal decision rules are $c_t = (1-\alpha\beta)z_t k_t^\alpha$ and $k_{t+1} = \alpha\beta z_t k_t^\alpha$. As in AFV, productivity follows a Markov chain with support $z_t \in \{0.9792, 0.9896, 1.0000, 1.0106, 1.0212\}$ and transition matrix:

$$\Pi = \begin{pmatrix} 0.9727 & 0.0273 & 0 & 0 & 0 \\ 0.0041 & 0.9806 & 0.0153 & 0 & 0 \\ 0 & 0.0082 & 0.9837 & 0.0082 & 0 \\ 0 & 0 & 0.0153 & 0.9806 & 0.0041 \\ 0 & 0 & 0 & 0.0273 & 0.9727 \end{pmatrix},$$

which approximates an autoregressive process of order 1 for $\log z_t$.

AFV formulate the model as a Bellman equation with value function $v(k, z)$, following the recursive notation in which $'$ denotes a next-period variable:

$$v(k, z) = \max_{k'} \; \{\ln(zk^\alpha - k') + \beta\mathbb{E}[v(k', z') \mid z]\}\,.$$

Next, AFV discretize the capital stock into 17,820 uniformly spaced points over $[0.5\bar{k}, 1.5\bar{k}]$, where $\bar{k}$ is the steady-state capital. The problem is solved using value function iteration (VFI). In the `C++` implementation, AFV report a computation time of 0.73 seconds.[19]

We cannot implement VFI on current QAs. State-of-the-art machines output only 0.625 kB of classical information per anneal, whereas the lookup table in the DPP above requires 89,100 floating-point numbers, which is at least 285 times more classical information using half-precision.

To circumvent this constraint, we adopt parametric dynamic programming (PDP), which allows the solution to be encoded using a small number of bits.[20] Even in the absence of the quantum-to-classical bottleneck, PDP would remain preferable, as our goal is to scale to large problem instances on a QA. Even if an exponential amount of classical information could be read out, storing and processing it classically would be infeasible.

In practice, PDP often yields faster solutions than VFI and scales better with the dimensionality of the state space. However, it is typically less stable (see, e.g., Taylor and Uhlig,

[19] This is reduced to approximately 0.26 seconds on a 2025-vintage PC.

[20] We evaluate the value function at all productivity nodes and at a number of capital nodes proportional to the square root of the number used in AFV. This already exceeds the number required to identify the value function parameters. The parameters in the quantum algorithms are chosen from a set of 2^{14} to 2^{20} elements, depending on the algorithm.

1990, Benítez-Silva et al., 2000, and Sweeting, 2013) and does not have a provably lower time complexity. To ensure full compatibility of results, we first solve the PDP version of the problem on a classical computer, establishing a clear benchmark for comparison.

5 Methods

We now describe how to set up the PDP problem associated with our RBC model, translate it into a PBF, quadratize the PBF into a QUBO, and execute the QUBO on a QA.

5.1 Setting Up the PDP

We use a parametric policy iteration (PPI) formulation that follows Benítez-Silva et al. (2000), where both the policy and value functions take parametric forms to ensure a space-efficient solution. In applied work, parametric approximation methods have typically yielded faster, though somewhat less stable, results than discrete policy iteration (Benítez-Silva et al., 2000).

Brock and Mirman (1972) showed that, when $\delta = 1$, the value function in our model is linear in the log of output (y), and that the policy rules are linear in output. Building on this insight, we assume the following functional forms for the value function and the policy rules for capital and consumption:

$$v(k, z) = x_2 + x_3 \ln(y) \tag{1}$$

$$k' = x_1 y \tag{2}$$

$$c = (1 - x_1)y. \tag{3}$$

Here, x_2 and x_3 are parameters of the value function to be determined, while x_1 is shared by both policy functions. Since output y must be allocated between consumption and capital, it follows that $c = y - k' = (1 - x_1)y$, consistent with the equations above. The functional forms (1)-(3) are chosen to align with the structure of the DPP and are not dictated by any requirement of quantum annealers.

We then substitute these expressions into the Bellman equation, yielding:

$$v(k, z) = \max_{x_1} \{\ln[(1 - x_1)y] + \beta \mathbb{E}[v(x_1 y, z')|z]\}. \tag{4}$$

In more complex models, richer functional approximations may be necessary (e.g., when the theoretical functional form of the Bellman equation or the policy function is unknown). Neural networks, for instance, can flexibly approximate the value function and policy rules (Fernández-Villaverde et al., 2023). A generic neural network representation of the value function could

be:

$$v(k,z) = x_0 + \sum_{i=1}^{n} x_i \phi(k, z, x_{i,j}),$$

where $\phi(\cdot)$ is a non-linear activation function and $\{x_{i,j}, x_i\}$ for all $\{i, j\}$ are the network weights. While this formulation introduces additional parameters, it is a straightforward extension of our approach, well-suited to problems with higher dimensionality or non-linearity.

PPI alternates between policy improvement and policy valuation steps. The policy improvement step uses equation (1) and is defined as:

$$x_1^* = \underset{x_1}{\operatorname{argmin}} \left\{ -\ln[(1 - x_1)y] - \beta \mathbb{E}[\bar{x}_2 + \bar{x}_3 \ln(y') \mid z] \right\}.$$

Here, $\bar{x}_2$ and $\bar{x}_3$ are treated as fixed parameters during this step, as indicated by the overbars.

Rewriting equation (4), we obtain a residual function. The policy valuation step then minimizes the squared residuals by choosing x_2 and x_3:

$$x_2^*, x_3^* = \underset{x_2, x_3}{\operatorname{argmin}} \left\{ \ln[(1 - \bar{x}_1)y] + \beta \mathbb{E}[x_2 + x_3 \ln(y') \mid z] - x_2 - x_3 \ln(y) \right\}^2.$$

To streamline notation, we denote the objective function for the policy improvement step as $g_p(x_1; \bar{x}_2, \bar{x}_3)$ and the objective function for the policy valuation step as $g_v(x_2, x_3; \bar{x}_1)$. Starting from an initial parameter vector $\{x_1, x_2, x_3\}$, PPI iterates over $g_p(x_1; \bar{x}_2, \bar{x}_3)$ and $g_v(x_2, x_3; \bar{x}_1)$ until convergence is reached. Algorithm 1 summarizes these steps.

Algorithm 1: Parametric Policy Iteration

1. Select a parametric form for the policy and value functions.
2. Initialize the parameters of the policy function, x_1, and value function, $\{x_2, x_3\}$.
3. Update the policy function parameter: $x_1^* = \underset{x_1}{\operatorname{argmin}}\{g_p(x_1; \bar{x}_2, \bar{x}_3)\}$.
4. Update the value function parameters: $x_2^*, x_3^* = \underset{x_2, x_3}{\operatorname{argmin}}\{g_v(x_2, x_3; \bar{x}_1)\}$
5. Repeat steps 3 and 4 until the convergence criterion is satisfied.

Implementing Algorithm 1 on a classical computer is straightforward. Average terminal errors across x_1, x_2, and x_3 from 10 executions of the algorithm are 1.37%, 0.77%, and 3.74%, respectively. Convergence is achieved in 0.58 seconds, which is faster than the 0.73 seconds reported for the `C++` benchmark VFI solution in AFV, but slower than the 0.26 seconds obtained for the same VFI solution on 2025-era hardware. Figure C.2 in Appendix C shows the average results from 10 executions of Algorithm 1 on a classical machine.

Thus, classical PPI does not yield faster convergence than classical VFI for the problem at hand. The AFV benchmarks cited later in the paper are therefore not biased in favor of the QA

due to our use of PPI instead of VFI. As we will demonstrate, a QA delivers an improvement over these classical benchmarks by an order of magnitude—regardless of whether the classical baseline uses PPI or VFI.

Programming a quantum annealer to execute Algorithm 1 is more involved, as we must first express it as a QUBO problem. This transformation requires an intermediate step: rewriting the problem as a pseudo-Boolean optimization (PBO) problem. The following two subsections detail how we implement these steps.

5.2 Translating the PDP into a PBO

Formulating the PDP problem as a pseudo-Boolean optimization (PBO) problem entails first mapping $\{x_1, x_2, x_3\}$ to binary variables. The simplest approach would discretize $\{x_1, x_2, x_3\}$, assign each node a binary variable, and impose the condition that the sum of the binary variables equals one. However, this strategy is inefficient and quickly exhausts the QA's computational resources.

An alternative strategy, which we adopt for $\{x_2, x_3\}$, expresses each variable using a standard binary encoding. This method represents N nodes x_ν in the state space with $\lceil \log_2(N) \rceil$ qubits:

$$x_\nu = s_\nu \sum_{j_\nu=0}^{J_\nu} 2^{j_\nu} x_{\nu,j_\nu}.$$

Here, s_ν is a scaling factor for variable x_ν, where $\nu \in \{1, 2, 3\}$, such that $x_\nu \in [0, s_\nu(2^{J_\nu+1} - 1)]$.

As an example, take $\nu = 2$, $s_2 = 1$, and $J_2 = 2$. If $x_{2,0} = 1$, $x_{2,1} = 0$, and $x_{2,2} = 1$, then $x_2 = 2^0 \cdot 1 + 2^1 \cdot 0 + 2^2 \cdot 1 = 5$. To cap the maximum value at m, we set the scaling factor to $s_2 = \frac{m}{2^{J_2+1}-1}$. Increasing J_2 improves the precision of x_2 for a fixed s_2. With only 19 qubits, we can construct a grid with over 10^6 nodes without introducing quadratic terms or constraints.[21]

A second, more substantive challenge involves representing the terms $\ln(x_1)$ and $\ln(1 - x_1)$ from Algorithm 1 on a QA. A naive solution might treat $\ln(x_1)$ and $\ln(1 - x_1)$ as separate variables linked by a constraint. But this is inefficient, as it requires two sets of binary variables and a quadratic constraint to enforce inversion, inflating connectivity requirements and network size.

Instead, we represent x_1 using a single set of binary variables and use two separate sets of polynomial coefficients to approximate:

$$\ln(x_1) \approx \left[a_0 + \sum_{j_1=0}^{J_1} a_1 2^{j_1} x_{1,j_1} + a_2 \sum_{j_1=0}^{J_1} \sum_{i_1=0}^{J_1} 2^{j_1+i_1} x_{1,j_1} x_{1,i_1} \right] \tag{5}$$

[21] The size difference between the coefficients of the smallest and largest terms may still pose challenges for a QA. It is thus important to evaluate how much precision is necessary for a given problem.

and:

$$\ln(1-x_1) \approx \left[\tilde{a}_0 + \tilde{a}_1 \sum_{j_1=0}^{J_1} 2^{j_1} x_{1,j_1}\right],$$

where $\{a_0, a_1, a_2\}$ and $\{\tilde{a}_0, \tilde{a}_1\}$ are the sets of coefficients.[22] Notice that these approximations are not Taylor expansions around a specific point. Rather, they are binary representations over the domain of permissible x_1 values that meet the requirements of a QUBO.

We can exploit symmetry and the fact that binary variables satisfy $x^2 = x$ to rewrite equation (5) as:

$$\ln(x_1) \approx \left[a_0 + \sum_{j_1=0}^{J_1}(a_1 2^{j_1} + a_2 2^{2j_1}) x_{1,j_1} + 2a_2 \sum_{j_1=0}^{J_1} \sum_{i_1<j_1} 2^{j_1+i_1} x_{1,j_1} x_{1,i_1}\right].$$

This form reduces the number of auxiliary variables needed to quadratize cubic and higher-order terms.[23] Furthermore, the use of quadratic terms does not require the introduction of additional binary variables. Hence, x_1, x_2, and x_3 can be represented using the same number of binary variables. Quadratic terms, however, require connectivity between qubits, a limited computational resource in QAs.

To fully characterize the PBO, we must re-express each of the terms in g_p and g_v as products and sums of binary variables. However, it will be convenient first to simplify their expressions. We start by substituting in the definition of y' and collecting monomial terms:

$$g_p(x_1; \bar{x}_2, \bar{x}_3) = \{-\ln(y) - \ln(1-x_1) - \beta\bar{x}_2 - \beta(\mathbb{E}[\ln(z')|z] + \alpha\ln(y))\bar{x}_3 - \alpha\beta\ln(x_1)\bar{x}_3\}.$$

Since y, $\bar{x}_2$, and $\bar{x}_3$ are constants in g_p, we eliminate the x_2 term and minimize the simplified version of the objective function:

$$g_{p'}(x_1; \bar{x}_2, \bar{x}_3) = \{-\ln(1-x_1) - \alpha\beta\bar{x}_3\ln(x_1)\}.$$

By eliminating the x_2 term, this simplification facilitates the construction of algorithms that can be executed entirely on a QPU. We will return to this point below.

We can now write down a PBO for the policy improvement step. Since $a_0 < 0$, $\tilde{a}_0 < 0$, and $\bar{x}_3 > 0$, we have $-(\tilde{a}_0 + \alpha\beta a_0\bar{x}_3) > 0$. Consequently, removing the constant term might allow for negative energy states (loss function evaluations), which are problematic for one of

[22] We have $[a_0, a_1, a_2] = [-0.10905, 0.57570, -1.38445]$ and $[\tilde{a}_0, \tilde{a}_1] = [-0.22278, -0.28375]$.

[23] This holds for QUBO models. In Ising models, where $x \in \{-1, 1\}$, the substitution does not apply.

the solution algorithms we introduce in Section 6. Thus, we retain the constant and write:

$$x^*_{1,0},...,x^*_{1,J_1} = \underset{x_{1,0},...,x_{1,J_1}}{\operatorname{argmin}} \Bigg\{ -(\tilde{a}_0 + a_0\alpha\beta\bar{x}_3) - \sum_{j_1=0}^{J_1} \Bigg[(\tilde{a}_1 + a_1\alpha\beta\bar{x}_3)2^{j_1} + a_2\alpha\beta\bar{x}_3 2^{2j_1} \Bigg] x_{1,j_1} - 2a_2\alpha\beta\bar{x}_3 \sum_{j_1=0}^{J_1} \sum_{i_1<j_1} 2^{j_1+i_1} x_{1,j_1}x_{1,i_1} \Bigg\}. \tag{6}$$

We next convert the policy valuation step into a PBO, starting with the expression:

$$g_v(x_2, x_3; \bar{x}_1) = \{\ln(y) + \ln(1-\bar{x}_1) - (1-\beta)x_2 + \zeta x_3 + \alpha\beta\ln(\bar{x}_1)x_3\}^2,$$

where we have defined $\zeta = \beta\mathbb{E}[\ln(z')|z] + (\alpha\beta - 1)\ln(y)$.

We now expand g_v:

$$\begin{aligned} g_v(x_2, x_3; \bar{x}_1) = \Bigg\{ & \gamma_0 + \gamma_1(\bar{x}_1) - \underbrace{2(1-\beta)(\ln(y) + \ln(1-\bar{x}_1))}_{\gamma_2(\bar{x}_1)} x_2 \\ & + \underbrace{2\Big[\ln(y)(\zeta + \alpha\beta\ln(\bar{x}_1)) + \ln(1-\bar{x}_1)(\zeta + \alpha\beta\ln(\bar{x}_1))\Big]}_{\gamma_3(\bar{x}_1)} x_3 \\ & - \underbrace{2(1-\beta)\Big[\zeta + \alpha\beta\ln(\bar{x}_1)\Big]}_{\gamma_{23}(\bar{x}_1)} x_2x_3 + \underbrace{(1-\beta)^2}_{\gamma_{22}} x_2^2 \\ & + \underbrace{\Big[\zeta^2 + 2\zeta\alpha\beta\ln(\bar{x}_1) + \alpha^2\beta^2\ln(\bar{x}_1)^2\Big]}_{\gamma_{33}(\bar{x}_1)} x_3^2 \Bigg\}. \end{aligned} \tag{7}$$

Notice that $\gamma_0 = \ln(y)^2 - 2(1-\beta)\ln(y)$ and $\gamma_1(\bar{x}_1) = 2\ln(y)\ln(1-\bar{x}_1) + \ln(1-\bar{x}_1)^2$. We enclose $\bar{x}_1$ in parentheses to indicate that the constant term γ_1 depends on $\bar{x}_1$ and will change after each policy improvement step.

Using the constants defined above, we rewrite equation (7) more compactly as:

$$g_v(x_2, x_3; \bar{x}_1) = \Big\{\gamma_0 + \gamma_1(\bar{x}_1) - \gamma_2(\bar{x}_1)x_2 + \gamma_3(\bar{x}_1)x_3 - \gamma_{23}(\bar{x}_1)x_2x_3 + \gamma_{22}x_2^2 + \gamma_{33}(\bar{x}_1)x_3^2\Big\}.$$

Finally, we define the PBO for the policy valuation step:

$$
\begin{aligned}
x^*_{2,0},...,x^*_{2,J_2},x^*_{3,0},...,x^*_{3,J_3} = \underset{x_{2,0},...,x_{2,J_2},x_{3,0},...,x_{3,J_3}}{\operatorname{argmin}} \Bigg\{ & \gamma_0 + \gamma_1(\bar{x}_1) \\
& - \gamma_2(\bar{x}_1)s_2 \sum_{j_2=0}^{J_2} 2^{j_2} x_{2,j_2} + \gamma_3(\bar{x}_1)s_3 \sum_{j_3=0}^{J_3} 2^{j_3} x_{3,j_3} \\
& - \gamma_{23}(\bar{x}_1)s_2 s_3 \sum_{j_2=0}^{J_2} \sum_{j_3=0}^{J_3} 2^{j_2+j_3} x_{2,j_2} x_{3,j_3} \\
& + \gamma_{22} s_2^2 \left(\sum_{j_2=0}^{J_2} 2^{2*j_2} x_{2,j_2} + \sum_{j_2=0}^{J_2} \sum_{i_2<j_2} 2^{j_2+i_2+1} x_{2,j_2} x_{2,i_2} \right) \\
& + \gamma_{33}(\bar{x}_1)s_3^2 \left(\sum_{j_3=0}^{J_3} 2^{2*j_3} x_{3,j_3} + \sum_{j_3=0}^{J_3} \sum_{i_3<j_3} 2^{j_2+i_2+1} x_{3,j_3} x_{3,i_3} \right) \Bigg\}.
\end{aligned} \tag{8}
$$

5.3 Translating the PBF to a QUBO

Converting a PBF into a QUBO entails quadratizing all cubic and higher-order terms. This reduces products of three or more binary variables into expressions containing no more than two-variable products, without introducing approximation error. The policy improvement and valuation steps, as defined in equations (6) and (8), do not include such terms. Consequently, no quadratization is needed if x_1 enters g_v as a constant and x_2 and x_3 enter g_p as constants, as they do in the hybrid algorithm we propose in Section 6.

If, however, we want to reduce computational overhead by solving the problem entirely on the QPU, as we also do in Section 6, then we will need to introduce higher-order terms. Subsection 3.3.3 explained how to quadratize such terms. We also give two examples here in the context of our problem.

We first consider the case where all terms in g_p are multiplied by the same binary variable, x_p. This will transform the quadratic terms into cubic ones in the last summation:

$$
-\sum_{j_1=0}^{J_1} \sum_{i_1<j_1} 2a_2\alpha\beta\bar{x}_3 2^{j_1+i_1} x_{1,j_1} x_{1,i_1} x_p.
$$

Since $a_2\alpha\beta\bar{x}_3 2^{j_1+i_1} > 0$ for all $\bar{x}_3$, j_1, and i_1, each term in the summation is negative. Thus, we can use the NTR method described in Appendix B to quadratize the expression using one auxiliary variable, x_{a,i_1,j_1}, per term, yielding:

$$
-\sum_{j_1=0}^{J_1} \sum_{i_1<j_1} 2a_2\alpha\beta\bar{x}_3 2^{j_1+i_1} x_{a,i_1,j_1}(2 - x_{1,j_1} - x_{1,i_1} - x_p).
$$

This step adds a total of $\frac{J_1(J_1-1)}{2}$ auxiliary variables to the policy valuation step.

Now consider the case where x_1 does not enter g_v as a constant. This occurs whenever we use an algorithm solved exclusively on the QPU and, thus, cannot treat x_1 as a fixed parameter. The term $2\zeta\alpha\beta\ln(x_1)x_3^2$, for example, becomes:

$$2\zeta\alpha\beta ln(x_1)x_3^2 = 2\zeta\alpha\beta s_1 s_3^2 \Bigg\{ a_0 \sum_{j_3=0}^{J_3}\sum_{i_3=0}^{J_3} 2^{j_3+i_3} x_{3,j_3}x_{3,i_3} + a_1 \sum_{j_1=0}^{J_1}\sum_{j_3=0}^{J_3}\sum_{i_3=0}^{J_3} 2^{j_1+j_3+i_3} x_{1,j_1}x_{3,j_3}x_{3,i_3} + a_2 \sum_{j_1=0}^{J_1}\sum_{i_1=0}^{J_1}\sum_{j_3=0}^{J_3}\sum_{i_3=0}^{J_3} 2^{j_1+i_1+j_3+i_3} x_{1,j_1}x_{1,i_1}x_{3,j_3}x_{3,i_3} \Bigg\}.$$

The final two summations consist entirely of cubic or higher-order terms and, thus, require quadratization.[24] All terms are positive, which rules out the use of the NTR method for reduction. Instead, we apply the positive term reduction (PTR) method from Boros and Gruber (2014) and focus specifically on the final summation, which contains quartic terms. Each term requires two auxiliary variables:

$$2\zeta\alpha\beta s_1 s_3^2 \Bigg[\sum_{j_1=0}^{J_1}\sum_{i_1=0}^{J_1}\sum_{j_3=0}^{J_3}\sum_{i_3=0}^{J_3} x_{a_1,j_1,i_1,j_3,i_3}(2 + x_{1,j_1} - x_{1,i_1} - x_{3,j_3} - x_{3,i_3}) + x_{a_2,j_1,i_1,j_3,i_3}(1 + x_{1,i_1} - x_{1,j_1} - x_{3,j_3} - x_{3,i_3}) + x_{3,j_3}x_{3,i_3} \Bigg].$$

We can also further reduce the number of auxiliary variables by exploiting symmetry and the properties of binary variables, as we have done previously.

6 Results

So far, we have explained how to express our PPI problem as separate QUBO problems for the policy and value function updates; however, we are still not quite ready to solve our problem on a QA because Algorithm 1 alternates between policy and value function updates—an iterative structure QAs cannot handle natively.

To get around this problem, we proceed in four cumulative steps:

1. Subsection 6.1 runs the QUBO version of PPI, Algorithm 2, on a classical computer, iterating over value and policy function updates. This step allows us to compare the QUBO version of PPI with the original PPI in Algorithm 1, which we already ran on a classical computer (Figure C.2).

[24] Recall that a QUBO does not permit cubic or higher-order terms. Quadratization reduces such terms to lower-order terms without introducing approximation error.

2. Subsection 6.2 proposes a new quantum–classical hybrid algorithm, Algorithm 3, where we run one part of the PPI on a classical computer and the rest on a QA. The use of hybrid algorithms has become popular in the literature (Hull et al., 2020).

3. Subsection 6.3 considers a fully quantum algorithm, Algorithm 4, that exploits reverse and inhomogeneous annealing to perform a complete iteration of policy and value function updates within an anneal. It also eliminates the need to re-program the QPU and iterates across anneals.

4. Subsection 6.4 modifies Algorithm 4 to perform multiple iterations within a single anneal and delivers Algorithm 5, which yields a candidate solution from each anneal.

Algorithms 4 and 5 are the most innovative parts of our paper from a technical perspective. We now go over each step in order.

6.1 Classical Combinatorial PPI

First, we solve the PPI problem with a classical combinatorial algorithm that performs alternating iterations of policy improvement and valuation steps. This algorithm is the combinatorial version of Algorithm 1. The policy improvement step is solved analytically, whereas the policy valuation step is specified as a QUBO and solved exactly by searching over the discretized value function parameters. The details are given in Algorithm 2.

Algorithm 2: Classical Combinatorial Parametric Policy Iteration

1 Initialize the value function parameters, x_2 and x_3.
2 Compute x_1 analytically.
3 Given the QUBO and x_1, perform the policy valuation step by identifying the values of $\{x_{2,0}, ..., x_{2,J_2}\}$ and $\{x_{3,0}, .., x_{3,J_3}\}$ that minimize the loss function.
4 Map the solution to value function parameters: $\{x_{2,0}, ..., x_{2,J_2}\} \rightarrow x_2$ and $\{x_{3,0}, .., x_{3,J_3}\} \rightarrow x_3$.
5 Repeat steps 2-4 until the convergence criterion is satisfied.

Using an exact solution for x_1 provides us with a benchmark that has three advantages. First, it eliminates errors from the policy valuation step, yielding an approximate measure of errors arising from discretization. Second, it provides an upper bound for acceptable candidate solution run times since it is always better to use an exact solution when the execution time is lower than the alternatives. Third, it helps evaluate how many iterations will be needed to achieve convergence in the hybrid and quantum algorithms.

We parameterize the algorithm by setting $J_2 = J_3 = 9$, which allows x_2 and x_3 to each take on 2^{10} discrete values, requiring a search over 2^{20} states on each policy valuation step.[25] We select scaling factors $s_2 = -0.035$ and $s_3 = 0.003$ to bound the value function parameters, $x_2 \in [0, 2x_2^*]$ and $x_3 \in [0, 2x_3^*]$, where x_2^* and x_3^* correspond to the true parameter values.

We execute the algorithm 10 times and calculate the average parameter error and execution time. After two iterations, the average terminal errors for x_1, x_2, and x_3 are 0.04%, 1.24%, and 0.14%, respectively. Other parameter configurations yield lower errors for individual parameters but at the expense of higher errors for other parameters and a higher loss. Convergence is achieved after 3.33 seconds (two iterations) on average, which is slower than the VFI solution in AFV and Algorithm 1.

All run time comparisons here and below exclude initial pre-processing and terminal post-processing times. Since the benchmark we select is competitive and can be solved classically in under one second (see the results in AFV), we restrict the comparisons to tasks where there are substantive differences across algorithms. Removing fixed and polynomial time computational overhead common to all algorithms simplifies this comparison.

The classical computational steps are performed on a Linux machine with an 8-core Intel Xeon 2.00GHz processor and 50GBs RAM. The classical algorithms and classical components of hybrid algorithms were programmed in Python. The quantum algorithms and quantum components of classical algorithms were executed on D-Wave's *Advantage* System 6.1, which has 5616 qubits, arranged in a lattice of 16×16 tiles known as a *Pegasus* (P16) graph. We have released code that can be used to reproduce the results in the paper.[26]

6.2 Hybrid Quantum-Classical Algorithm

We next approach the PPI problem with a hybrid algorithm that performs alternating iterations of policy improvement and valuation steps. The policy valuation step is run on the QA, while the rest of the algorithm is executed classically. The details are given in Algorithm 3. Notice that $[C]$ indicates a step is performed classically and $[Q]$ on a QA.

As before, we set $J_2 = 9$, $J_3 = 9$, $s_2 = -0.035$, and $s_3 = 0.003$, and use 100 anneals, each with a duration of $20\mu s$ (microseconds). We then select the 10% of anneals with the lowest energy levels and terminate the process after two iterations. The average energy level (loss) of the system tends to decline substantially over the first two iterations; however, subsequent steps have considerably smaller impacts on the loss. This aligns well with the classical results.

Notice that the hybrid algorithm contains quantum components that are affected by *true*

[25] We choose values that are sufficient for achieving convergence and are consistent with a high-precision solution.

[26] See the GitHub repository at https://github.com/ijh85/quantum-dynamic-programming, which contains code that can be used to verify the results in the paper.

Algorithm 3: Hybrid Parametric Policy Iteration

1 [C] Initialize the value function parameters, x_2 and x_3.
2 [C] Compute x_1 analytically.
3 [Q] Given the QUBO and x_1, perform the policy valuation step, yielding $\{x_{2,0}, ..., x_{2,J_2}\}$ and $\{x_{3,0}, .., x_{3,J_3}\}$ for each anneal.
4 [C] Map the output of each anneal to value function parameters: $\{x_{2,0}, ..., x_{2,J_2}\} \to x_2$ and $\{x_{3,0}, .., x_{3,J_3}\} \to x_3$.
5 [C] Compute the mean values of x_2 and x_3 for the lowest energy level anneals.
6 Repeat steps 2-5 until the convergence criterion is satisfied.

randomness that arises in quantum systems. Consequently, the candidate solutions that the algorithm produces cannot be reproduced from an initial state. For this reason, we focus on the distribution of candidate solutions produced by 50 executions of the algorithm.

	x_1 Error %	x_2 Error %	x_3 Error %	QPU Total	QPU Prog.	Total Time
Mean	5.70	1.34	9.63	5.23E+04	3.19E+04	1.76E+06
25th Percentile	3.12	0.58	5.33	5.23E+04	3.19E+04	1.53E+06
75th Percentile	7.74	2.03	13.22	5.23E+04	3.19E+04	1.60E+06
Std. Dev.	3.99	0.90	6.00	0.0E+04	0.00E+04	0.78E+06

Table 2: The initial values are $x_1 = 0.5$, $x_2 = -0.5$, and $x_3 = 0.5$, which lie in the feasible range for each parameter and represent absolute deviations of 50% to 100% from their true values. This initialization does not apply to the external classical benchmarks or the purely quantum solutions in the paper. All times are reported in microseconds. Errors for the policy and value function parameters are expressed as absolute percentage deviations from their true values. Summary statistics are computed from 50 executions of the same hybrid program. The quantum part of each iteration uses 100 anneals, retaining the 10% with the lowest energy levels.

Table 2 reports the results. The error columns for x_1, x_2, and x_3 show the absolute percentage deviations from the true parameter values. The next three columns report total quantum processor time (QPU Total), quantum programming time (QPU Prog.), and total execution time (Total Time), including classical and quantum components.

The hybrid algorithm typically converges near $\{x_1^*, x_2^*, x_3^*\}$ in two iterations. On average, it completes in 1.76 seconds, which is faster than the classical combinatorial solution, but 2.4 times slower than the fastest `C++` VFI implementation in AFV, which ran in 0.73 seconds. Only 3% of the total time is spent on QPU computation. And just 39% of that on annealing. The remainder of the QPU time comes from the QPU programming step, incurred once per problem before a batch of anneals. Switching to a fully quantum algorithm and programming the QPU only once could reduce run time from 1.76 seconds to under 0.03 seconds.

While faster, the hybrid algorithm is less precise than the classical benchmarks. For example, the terminal value of x_3 deviates from x_3^* by 9.63% on average. This is consistent with prior findings that QAs yield fast, high-quality approximations but require more time for refinement

(Zintchenko et al., 2015; King et al., 2015). If greater precision is required, the QA output can warm-start a classical solver.

6.3 Multi-Anneal Quantum Algorithm

Motivated by the previous results, we construct an algorithm that eliminates the classical components and requires programming the QPU only once. More concretely, we specify a single QUBO and an annealing schedule that iteratively optimize the two components of the objective function. To do so, we introduce two recent computational strategies: reverse annealing and inhomogeneous annealing.

Reverse annealing initializes the QPU in a classical state rather than in a uniform superposition over the full state space. It then partially opens a superposition weighted toward that state. A full reversal is equivalent to standard forward annealing, whereas a small reversal favors solutions near the initial state. Reverse annealing is typically used to refine solutions by searching within a neighborhood (Pelofske et al., 2020). The size of the reversal determines the size of the neighborhood.[27]

Inhomogeneous annealing assigns offset values to each qubit. A positive offset indicates that a qubit should be annealed ahead of the global schedule; a negative value delays it. This technique has been applied to improve alignment between multiple qubits in the annealing process (Adame and McMahon, 2020).

We use both techniques not for local refinement but to facilitate iteration over two separate objective functions, diverging from standard usage. Algorithm 4 performs this iteration across anneals. Algorithm 5, introduced next, performs the iteration within a single anneal. In both cases, the QPU is programmed only once.

Algorithm 4: Quantum Parametric Policy Iteration

1 Quadratize the policy improvement and valuation PBFs separately, reconcile the resulting QUBOs, and then merge them.
2 Initialize the classical states of policy function variables, $\{x_{1,0}, ..., x_{1,J_1}\}$, and value function variables, $\{x_{2,0}, ..., x_{2,J_2}, x_{3,0}, ..., x_{3,J_3}\}$.
3 Initialize variables that activate and deactivate loss components, $x_p = 0$ and $x_v = 0$.
4 Set an inhomogeneous annealing schedule that first anneals $\{x_{1,0}, ..., x_{1,J_1}\}$ and x_p, and then anneals $\{x_{2,0}, ..., x_{2,J_2}, x_{3,0}, ..., x_{3,J_3}\}$ and x_v.
5 Set a reverse annealing schedule that applies to all qubits.
6 Perform $\mathcal{N}$ anneals without reinitializing the original classical state.
7 Select the results from the anneal with the lowest energy level.

[27] We use a full reversal to avoid restricting the search to a local neighborhood. Partial reversals could offer more stability but would limit the search space.

Algorithm 4 describes the multi-anneal quantum routine. We parameterize it by setting $J_1 = J_2 = J_3 = 6$, so x_1, x_2, and x_3 each take on 2^7 discrete values.[28]

Step 1 uses a non-standard approach to quadratize the PBF into a QUBO. Specifically, it quadratizes $x_p g_p$ and $x_v g_v$ separately, reconciles the two resulting QUBOs, and merges them. This method ensures that qubits not relevant to a given component remain inactive and are annealed with the correct group.

Steps 2 and 3 initialize the QA in a classical state where both objective components are inactive (i.e., $x_p = 0$ and $x_v = 0$). In Steps 4 and 5, a reverse anneal opens a superposition for $\{x_{1,0}, ..., x_{1,J_1}\}$ and x_p, allowing for optimization of the policy function and its parameters, while keeping the value function and its parameters inactive. As the anneal progresses through its schedule, the qubit offsets specify that a superposition should be opened for $\{x_{2,0}, ..., x_{2,J_2}, x_{3,0}, ..., x_{3,J_3}\}$ and x_v, enabling optimization over the value function and with value function parameters. The modified objective function is then given by:

$$g = x_p g_p(x_1; \bar{x}_2, \bar{x}_3) + x_v g_v(x_2, x_3; \bar{x}_1)$$

Two additional details complete the algorithm. First, the QA should not reinitialize after each anneal. This ensures that each anneal in the sequence begins in the terminal classical state of the previous one, allowing repeated policy improvement and valuation steps.

Second, the problem setup and annealing schedule must ensure that $x_p = 0$ and $x_v = 0$ emerge as the terminal classical states. This is necessary because x_p must converge to zero to deactivate the policy function component during the valuation step. It also guarantees that subsequent anneals begin with both x_p and x_v set to zero. This condition can be enforced by adding a positive bias term to one component of the objective function and increasing its magnitude until anneals consistently return $x_p = 0$ and $x_v = 0$.

Figure 4 provides a stylized depiction of the annealing schedule. The policy function parameters and their objective component are annealed first (dashed line), followed by the value function and its parameters (solid line). In both cases, parameters begin in a classical state and are then reverse and forward annealed. The annealing activity at each step is illustrated as a weighted graph in Figure E.3 of Online Appendix E.

Table 3 reports summary statistics from 50 executions of Algorithm 4, each consisting of 20 anneals. For each QPU execution, we select the parameter values associated with the lowest-energy anneals for both objective function components.[29] The total QPU computation time

[28] We reduce J_2 and J_3 to fit the full multi-objective problem on the annealer, which requires more connectivity and space to represent x_1. This lowers the precision of x_2 and x_3, possibly increasing error size, but it does not affect execution time.

[29] The terminal energy levels are not a reliable comparison metric, as our objective is zeroed out across the anneal. We instead reconstruct each component of the objective function.

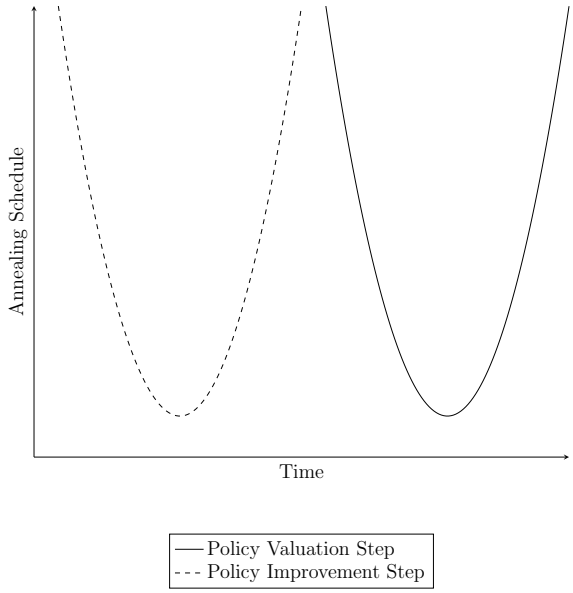


Figure 4: Stylized depiction of the annealing schedule.

drops to 0.0217 seconds, roughly 3% of the 0.73 seconds for the `C++` VFI solution in AFV and just 0.66% of the time required by the classical combinatorial algorithm in Subsection 6.1.

	x_1 Error %	x_2 Error %	x_3 Error %	QPU Total	QPU Programming Time
Mean	12.13	5.82	14.50	2.17E+04	1.60E+04
25th Percentile	9.62	3.33	7.40	2.17E+04	1.60E+04
75th Percentile	14.82	7.21	20.22	2.17E+04	1.60E+04
Standard Deviation	3.58	3.62	9.26	0.00E+00	0.00E+00

Table 3: We use a reverse annealing parameter of 0.00, which corresponds to a 100% reversal. All times are given in microseconds. Summary statistics are reported from 50 QPU executions of the same program, each of which performs 20 anneals.

While the run time is reduced, the mean policy and value function parameter errors across the 50 QPU executions increase slightly relative to the hybrid algorithm. Moreover, iterating across anneals without resetting the QPU leads to high correlation among candidate solutions within each execution. To address this, we execute the QPU multiple times to obtain independent solution candidates.

In the next subsection, we propose a refinement that transforms each anneal into an independent candidate solution. This approach enables us to obtain high-quality results reliably while programming and executing the QPU only once.

6.4 One-Shot Quantum Algorithm

Next, we propose a *one-shot* quantum PPI algorithm, Algorithm 5. In contrast to Algorithm 4, Algorithm 5 allows for multiple iterations over the policy and value functions within a single anneal. Additionally, it produces terminal energy levels that can be directly compared across anneals, enabling the reliable selection of candidate solutions with the lowest associated losses. This comparison is not feasible under Algorithm 4, since x_p and x_v converge to zero during the anneal. Relative to Algorithm 4, Algorithm 5 has two drawbacks: it requires longer annealing schedules and longer pauses between anneals to reinitialize the quantum state.

Algorithm 5: One-Shot Quantum Parametric Policy Iteration

1 Quadratize the policy improvement and valuation PBFs separately, reconcile the resulting QUBOs, and then merge them.
2 Initialize random classical states for the policy and value function variables: $\{x_{1,0}, ..., x_{1,J_1}, x_{2,0}, ..., x_{2,J_2}, x_{3,0}, ..., x_{3,J_3}\}$.
3 Initialize (de-)activation variables in classical zero states: $x_v = x_p = 0$.
4 Set an inhomogeneous annealing schedule to activate the policy improvement (g_p) and policy valuation (g_v) functions in sequence, and to anneal their associated variables $\mathcal{C}$ times.
5 Set the global annealing schedule to perform a full reversal.
6 Perform $\mathcal{N}$ anneals and reinitialize the state after each.
7 Compute alternative measures of terminal state energy for each anneal.
8 Return average parameter values over the lowest energy anneals.

By allowing for iteration within the annealing step, we can perform a full reverse anneal and explore the entire state space, rather than restricting the search to a neighborhood around the terminal state of the previous anneal. As a result, the terminal state of each anneal can be treated as a candidate solution, not merely an intermediate iteration. Furthermore, since no information needs to be preserved across anneals, we may reinitialize the state each time to reduce correlation across candidate solutions.

Algorithm 5 includes two novel components: (1) an inhomogeneous reverse annealing schedule that allows for multiple iterations *within* a single anneal, and (2) a post-processing routine that improves solution quality. We discuss each component before presenting the results.

Annealing schedule. Figure 5 illustrates the annealing schedule used in the one-shot algorithm. The schedule includes reversals and is inhomogeneous. Qubits are split into two groups and annealed separately within each *cycle*—a subschedule where all qubits are annealed once. The figure shows three cycles, each repeating the same subschedule.

Our algorithm departs from the original conception of quantum annealing, which relied on a simple forward anneal (Farhi et al., 2000). Existing applications of reverse annealing typically

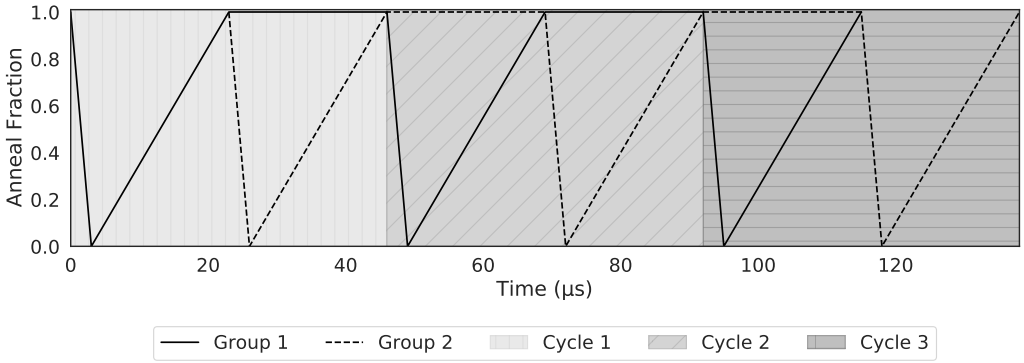


Figure 5: Experimental annealing schedules for three cycles. We adopt a reverse, inhomogeneous anneal. Group 1 contains qubits used in the policy improvement step. Group 2 contains qubits used in the policy valuation step. All times are given in microseconds (μs).

aim at refining candidate solutions (Pelofske et al., 2020), whereas we use reverse annealing to build iterative algorithms. To validate that Algorithm 5 functions as intended, we conduct two simpler experiments, with results reported in Online Appendix D.

The simulations show that cycling through repeated annealing subschedules emulates an iterative algorithm within a single anneal. This suggests that the annealing schedule in Algorithm 5, together with the (de-)activation mechanism, could support a one-shot quantum routine.

Finally, we clarify why initialization matters under full reversal. Figure 5 shows that qubits are divided into two groups and annealed separately in each cycle. One group remains in a classical state while the other is reversed into full superposition before the forward anneal. Initialization pins down the classical states of Group 2 during the first Group 1 anneal. But Group 1 is immediately reversed into a uniform superposition, so its initial classical state is not relevant.[30]

Post-processing. Algorithm 5 yields a candidate solution from each anneal, but the quality of these solutions often varies. This variation is reflected in the system's terminal energy level, which is typically used to select the best solution. As discussed earlier with Algorithm 4, however, these energy readings are unreliable in our case because the weights on different objective components shrink at different points during the annealing schedule.

To improve interpretability, we modify the post-processing routine. We first recompute the energy level in each terminal state without the de-activation qubits. We call this the *unadjusted*

[30]At the time of writing, it was not possible to assign separate annealing schedules to two qubit groups. Instead, the same schedule was applied with different offsets for each group. Hence, we could not start with a forward anneal for Group 1 and a reverse for Group 2. Nor could we begin with a forward anneal for both, as this would fail to deactivate the policy valuation component during the initial policy improvement step.

loss, which reflects total loss but applies no further transformation. Still, this measure is imperfect: it combines both components of the loss and is heavily influenced by errors in x_2. In practice, this concept of loss helps evaluate solution quality for x_2 but not for x_1 or x_3.

To address this, we construct the *minimum loss*, which isolates the effect of errors in each parameter. For example, to assess a candidate x_1^c, we compute $g_p(x_1^c; x_2^*, x_3^*)$, fixing other parameters at their true values. This reduces the variance caused by errors in other parameters and ensures that the correct parameter value (e.g., x_1^*) yields the lowest possible energy level.

In practice, the minimum loss is rarely feasible to compute because the true values of other parameters are unknown. Instead, we use the *adjusted loss*, which approximates the minimum loss and is computable. We drop x_2 terms from x_p (where they enter only additively) and compute the loss for each parameter, fixing the others at their average values from the lowest-energy anneals. This significantly reduces the variance in g_p from errors in x_2 and x_3.

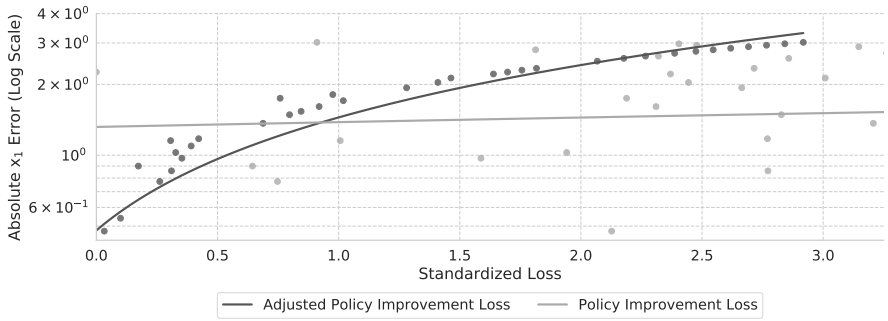
As a final step, we retain the subset of anneals with the lowest adjusted loss for each parameter and compute the parameter's mean value across those anneals. Table 4 compares the correlations between parameter errors and the three loss measures. As noted earlier, the unadjusted loss is strongly correlated with the error in x_2—nearly as strongly as the minimum loss. In contrast, its correlation with errors in x_1 and x_3 is weak, suggesting it is not a useful overall metric. The minimum loss, by contrast, is strongly correlated with parameter errors, confirming that eliminating noise from x_2 and x_3 allows us to identify anneals with useful information about x_1^*. The adjusted loss performs nearly as well, yielding correlation coefficients of 0.97 for x_1, 0.96 for x_2, and 0.97 for x_3.

	x_1 Error %	x_2 Error %	x_3 Error %
Unadjusted Loss	-0.01	0.92	0.05
Minimum Loss	0.97	0.96	0.97
Adjusted Loss	0.97	0.96	0.97

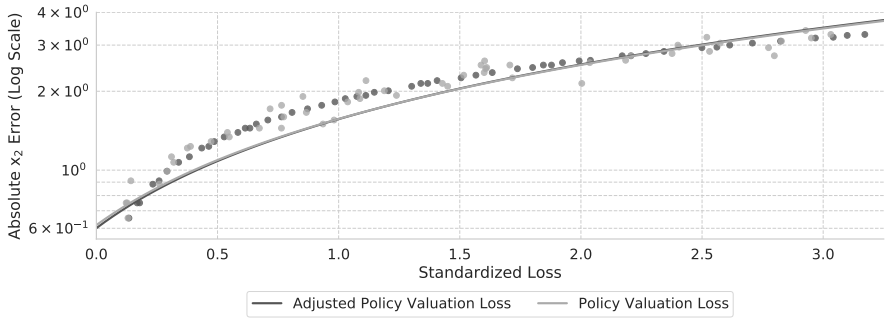
Table 4: Correlations between the percentage error in the solutions for the policy and value function parameters.

Figure 6 shows the benefits of using adjusted rather than unadjusted loss for post-processing. Each subfigure presents a binned scatterplot of loss values against absolute parameter errors. For x_2, both losses are similarly informative. But for x_1 and x_3, low adjusted loss values clearly identify high-quality solutions, whereas low unadjusted losses do not.

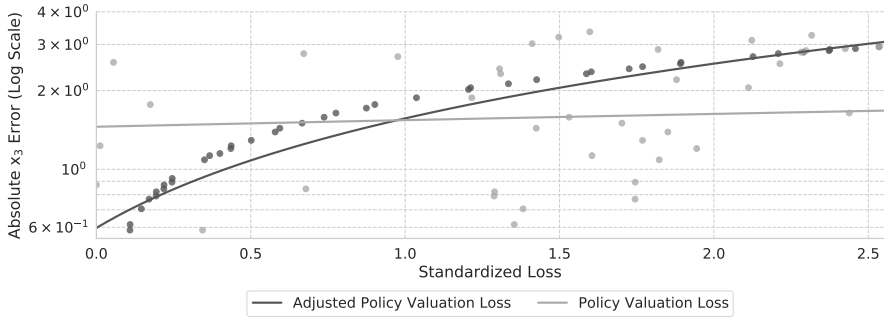
Results. Table 5 reports summary statistics from 50 QPU executions of Algorithm 5, each with 200 anneals. Within each anneal, we run three cycles of the annealing subschedule shown in Figure 5. The average policy and value function errors are lower than those from the multi-anneal quantum algorithm. Errors in x_1 and x_3 are also lower, on average, than those from



x_1 Error and Policy Improvement Loss



x_2 Error and Policy Valuation Loss



x_3 Error and Policy Valuation Loss

Figure 6: Parameter error in absolute percentage deviations with levels of the *unadjusted loss* and *adjusted loss*. This is visualized using a binned scatter plot and fitted line for the results from 200 anneals. The vertical axis uses a log scale because the absolute error typically increases sharply at low levels of the *adjusted loss*.

the hybrid algorithm. Total QPU execution time is 0.065 seconds, which is longer than the multi-anneal quantum algorithm, but only 3.7% of the hybrid algorithm's run time and 8.9% of the `C++` VFI implementation in AFV.

The run time rises relative to the multi-anneal algorithm for three reasons. First, the annealing schedule includes multiple cycles of reverse and forward annealing, raising the time per anneal from $23\mu s$ to $115\mu s$. Second, we reinitialize the classical state between anneals to improve the independence of candidate solutions, adding overhead. Third, we use 200 anneals to generate independent candidates instead of a few highly correlated ones.

	x_1 Error %	x_2 Error %	x_3 Error %	QPU Total	QPU Programming Time
Mean	2.98	1.71	4.52	6.52E+04	1.59E+04
25th Percentile	1.70	1.08	1.92	6.52E+04	1.59E+04
75th Percentile	2.49	2.01	3.98	6.52E+04	1.59E+04
Std. Dev.	3.48	0.87	5.47	0.00E+04	0.00E+04

Table 5: We use a reverse annealing parameter of 0.0, which corresponds to a 100% reversal. All times are given in microseconds. All anneals use three cycles. Summary statistics are reported from 50 QPU executions of the same program, each of which performs 200 anneals.

To give further intuition for the size of the errors, we run a 10-period simulation of the consumption path following a negative productivity shock. Figure 7 plots the paths from the analytical solution and the one-shot quantum algorithm. The difference is negligible in all periods and most visible right after the shock.

Figure 7: Consumption simulation paths generated from the true parameters and the one-shot solution parameters. In both cases, we simulate the 10-period consumption response to a negative productivity shock in the first period.

In summary, Algorithm 5 delivers a quantum solution that thoroughly explores the state space, lowers correlation across terminal states, and yields higher-quality solutions than the multi-anneal algorithm. It offers an order-of-magnitude speed-up over classical benchmarks, but remains slower than the multi-anneal approach.

7 Conclusion

We have introduced a set of algorithms for solving DPPs and other iterative computational tasks on a QA. Our approach can recover value and policy functions and is implementable

on current quantum hardware for small but non-trivial economic problems. By design, these algorithms avoid fundamental scaling limits, enabling application to larger models as hardware improves.

To demonstrate feasibility, we applied our algorithms to the RBC model in AFV. Using two fully quantum (non-hybrid) algorithms, we achieved run times close to the theoretical minimum for a QA. Although accuracy is lower than in classical methods, it remains high enough for typical RBC model applications.

We also discussed other potential DPP applications for quantum annealing. QAs could warm-start high-dimensional problems by providing parametric solutions to initialize classical algorithms. They could also estimate average or maximum errors over large grids, helping assess solution quality in regions far from the steady state, or detect where policy or value function features cause performance loss.

In summary, quantum computing shows promise for solving economic models that are otherwise intractable. While the technology is still maturing, QAs can already tackle problems of interest and compete with standard VFI and PPI benchmarks. The quantum computing literature on DPPs remains small, and much work is needed to explore its full potential. We hope this paper offers a foundation for future research in economics and finance and contributes a novel method for solving dynamic and iterative problems on a QA.

References

Adame, J. I. and McMahon, P. L. (2020). Inhomogeneous driving in quantum annealers can result in orders-of-magnitude improvements in performance. *Quantum Science and Technology*, 5(035011).

Albash, T. and Lidar, D. A. (2015). Decoherence in adiabatic quantum computation. *Physical Review A*, 91(6).

Albash, T., Rønnow, T., Troyer, M., and Lidar, D. (2015a). Reexamining classical and quantum models for the D-Wave one processor. *The European Physical Journal Special Topics*, 224(1):111–129.

Albash, T., Vinci, W., Mishra, A., Warburton, P. A., and Lidar, D. A. (2015b). Consistency tests of classical and quantum models for a quantum annealer. *Physical Review A*, 91:042314.

Ambainis, A., Balodis, K., Iraids, J., Kokainis, M., Prūsis, K., and Vihrovs, J. (2019). *Quantum Speedups for Exponential-Time Dynamic Programming Algorithms*, pages 1783–1793. ACM-SIAM.

Ambainis, A. and Regev, O. (2004). An elementary proof of the quantum adiabatic theorem. arXiv:quant-ph/0411152.

Aruoba, S. B. and Fernández-Villaverde, J. (2015). A comparison of programming languages in macroeconomics. *Journal of Economic Dynamics and Control*, 58:265–273.

Bapst, V., Foini, L., Krzakala, F., Semerjian, G., and Zamponi, F. (2013). The quantum adiabatic algorithm applied to random optimization problems: The quantum spin glass perspective. *Physics Reports*, 523(3):127–205.

Barahona, F. (1982). On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241–3253.

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

Benítez-Silva, H., Rust, J., Hitsch, G., Pauletto, G., and Hall, G. (2000). A comparison of discrete and parametric methods for continuous-state dynamic programming problems. Computing in Economics and Finance 2000, Society for Computational Economics.

Boixo, S., Rønnow, T. F., Isakov, S. V., Wang, Z., Wecker, D., Lidar, D. A., Martinis, J. M., and Troyer, M. (2014). Evidence for quantum annealing with more than one hundred qubits. *Nature Physics*, 10(3):218–224.

Boixo, S., Smelyanskiy, V. N., Shabani, A., Isakov, S. V., Dykman, M., Denchev, V. S., Amin, M. H., Smirnov, A. Y., Mohseni, M., and Neven, H. (2016). Computational multiqubit tunnelling in programmable quantum annealers. *Nature Communications*, 7(1):10327.

Boros, E. and Gruber, A. (2014). On quadratization of pseudo-Boolean functions. arXiv:1404.6538 [math.OC].

Boros, E. and Hammer, P. L. (2002). Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1):155–225.

Brock, W. and Mirman, L. (1972). Optimal economic growth and uncertainty: The discounted case. *Journal of Economic Theory*, 4(3):479–513.

Carrera Vazquez, A. and Woerner, S. (2021). Efficient state preparation for quantum amplitude estimation. *Physical Review Applied*, 15:034027.

Childs, A. M., Farhi, E., and Preskill, J. (2001). Robustness of adiabatic quantum computation. *Physical Review A*, 65(1).

Dattani, N. (2019). Quadratization in discrete optimization and quantum mechanics. arXiv:1901.04405 [quant-ph].

Dattani, N. and Chancellor, N. (2019). Embedding quadratization gadgets on Chimera and Pegasus graphs. arXiv:1901.07676 [quant-ph].

Dattani, N., Szalay, S., and Chancellor, N. (2019). Pegasus: The second connectivity graph for large-scale quantum annealing hardware. arXiv:1901.07636 [quant-ph].

Denchev, V. S., Boixo, S., Isakov, S. V., Ding, N., Babbush, R., Smelyanskiy, V., Martinis, J., and Neven, H. (2016). What is the computational value of finite-range tunneling? *Physical Review X*, 6:031015.

Egger, D., Gutierrez, R. G., Mestre, J., and Woerner, S. (2021). Credit risk analysis using quantum computers. *IEEE Transactions on Computers*, 70(12):2136–2145.

Farhi, E., Goldstone, J., and Gutmann, S. (2014). A quantum approximate optimization algorithm. arXiv:1411.4028 [quant-ph].

Farhi, E., Goldstone, J., Gutmann, S., and Sipser, M. (2000). Quantum computation by adiabatic evolution. arXiv:0001106 [quant-ph].

Fernández-Villaverde, J., Hurtado, S., and Nuño, G. (2023). Financial frictions and the wealth distribution. *Econometrica*, 91(3):869–901.

Freedman, D. and Drineas, P. (2005). Energy minimization via graph cuts: settling what is possible. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 939–946.

Ghysels, E. and Morgan, J. (2024). On quantum ambiguity and potential exponential computational speed-ups to solving dynamic asset pricing models. *SSRN Electronic Journal*.

Ghysels, E., Morgan, J., and Mohammadbagherpoor, H. (2023). Quantum computational algorithms for derivative pricing and credit risk in a regime switching economy. Discussion paper, UNC and IBM.

Glos, A., Kokainis, M., Mori, R., and Vihrovs, J. (2021). Quantum speedups for dynamic programming on n-dimensional lattice graphs. arXiv:2104.14384 [quant-ph].

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In Miller, G. L., editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 212–219. ACM.

Harrow, A., Hassidim, A., and Lloyd, S. (2009). Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15).

Heim, B., Brown, E. W., Wecker, D., and Troyer, M. (2017). Designing adiabatic quantum optimization: A case study for the traveling salesman problem. arXiv:1702.06248 [quant-ph].

Herman, D., Googin, C., Liu, X., Galda, A., Safro, I., Sun, Y., Pistoia, M., and Alexeev, Y. (2022). A survey of quantum computing for finance. arXiv:2201.02773 [quant-ph].

Hull, I., Sattath, O., Diamanti, E., and Wendin, G. (2020). Quantum technology for economists. arXiv:2012.04473 [econ.GN].

Imoto, T., Susa, Y., Miyazaki, R., Kadowaki, T., and Matsuzaki, Y. (2024). Universal quantum computation using quantum annealing with the transverse-field Ising Hamiltonian. arXiv:2402.19114 [quant-ph].

Ishikawa, H. (2014). Higher-order clique reduction without auxiliary variables. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1362–1369.

Kaneko, K., Miyamoto, K., Takeda, N., and Yoshino, K. (2022). Quantum pricing with a smile: implementation of local volatility model on quantum computer. *EPJ Quantum Technology*, 9(1):7.

Kieu, T. D. (2019). The travelling salesman problem and adiabatic quantum computation: an algorithm. *Quantum Information Processing*, 18(3).

King, J., Yarkoni, S., Nevisi, M. M., Hilton, J. P., and McGeoch, C. C. (2015). Benchmarking a quantum annealing processor with the time-to-target metric. arXiv:1508.05087 [quant-ph].

Kolmogorov, V. and Zabin, R. (2004). What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159.

Lucas, A. (2014). Ising formulations of many NP problems. *Frontiers in Physics*, 2.

Messiah, A. (1958). *Quantum mechanics*. Dover Publications.

Orús, R., Mugel, S., and Lizaso, E. (2019a). Forecasting financial crashes with quantum computing. *Physical Review A*, 99(6).

Orús, R., Mugel, S., and Lizaso, E. (2019b). Quantum computing for finance: Overview and prospects. *Reviews in Physics*, 4.

Ozfidan, I. et al. (2020). Demonstration of a nonstoquastic Hamiltonian in coupled superconducting flux qubits. *Physical Review Applied*, 13(3).

Pelofske, E., Hahn, G., and Djidjev, H. (2020). Advanced anneal paths for improved quantum annealing. arXiv:2009.05008 [quant-ph].

Shin, S. W., Smith, G., Smolin, J. A., and Vazirani, U. (2014). How "quantum" is the D-Wave machine? arXiv:1401.7087 [quant-ph].

Shor, P. W. (1997). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509.

Stamatopoulos, N., Egger, D. J., Sun, Y., Zoufal, C., Iten, R., Shen, N., and Woerner, S. (2020). Option pricing using quantum computers. *Quantum*, 4:291.

Sweeting, A. (2013). Dynamic product positioning in differentiated product markets: The effect of fees for musical performance rights on the commercial radio industry. *Econometrica*, 81(5):1763–1803.

Tanburn, R., Okada, E., and Dattani, N. (2015). Reducing multi-qubit interactions in adiabatic quantum computation without adding auxiliary qubits. arXiv:1508.04816 [quant-ph].

Taylor, J. B. and Uhlig, H. (1990). Solving nonlinear stochastic growth models: A comparison of alternative solution methods. *Journal of Business & Economic Statistics*, 8(1):1–17.

Venegas-Andraca, S. E., Cruz-Santos, W., McGeoch, C., and Lanzagorta, M. (2018). A cross-disciplinary introduction to quantum annealing-based algorithms. *Contemporary Physics*, 59(2):174–197.

Warren, R. H. (2019). Solving the traveling salesman problem on a quantum annealer. *SN Applied Sciences*, 2(1).

Woerner, S. and Egger, D. J. (2019). Quantum risk analysis. *npj Quantum Information*, 5(1).

Wootters, W. and Zurek, W. (1982). A single quantum cannot be cloned. *Nature*, 299(5886):802–803.

Zintchenko, I., Brown, E., and Troyer, M. (2015). Recent developments in quantum annealing. Unpublished Manuscript.

Online Appendix

A Transverse-field Ising Model

Here, we present the transverse-field Ising model—the quantum version of the classical Ising model introduced in Section 3. We begin with the standard initial state of a QA, given by the Hamiltonian:

$$\mathcal{H}_0 = -\sum_{i=0}^{N-1} \sigma_i^x,$$

where $\sigma_i^x = I \otimes ... \otimes \sigma^x \otimes ... \otimes I$, I is a 2x2 identity matrix, and σ^x is the Pauli X matrix:

$$\sigma^x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

where the i indicates that σ^X is located in the i^{th} position in the sequence of tensor products.

For the $N = 2$ case, for example, $\mathcal{H}_0$ would be specified as:

$$\mathcal{H}_0 = -(\sigma^x \otimes I + I \otimes \sigma^x) = \begin{bmatrix} 0 & -1 & -1 & 0 \\ -1 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 \\ 0 & -1 & -1 & 0 \end{bmatrix}.$$

The ground state corresponds to the eigenvector associated with the minimum eigenvalue of $\mathcal{H}_0$. In this case, the characteristic polynomial of the expression for $\mathcal{H}_0$ is $\lambda^4 - 4\lambda = 0$, yielding the eigenvalues $\lambda = \{-2, 2, 0, 0\}$ and eigenvectors v:

$$v = \left\{ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \\ 1 \\ 1 \end{bmatrix} \right\}.$$

The first eigenvector, $v_0 = [1, 1, 1, 1]$, corresponds to the lowest eigenvalue ($\lambda_0 = -2$) and defines the system's ground state—the global solution. After normalization, this vector represents a uniform superposition over all basis states.[31] With two qubits and two possible values each, there are four classical configurations. A QA is typically initialized in this state (for N qubits),

[31] A superposition is a linear combination of basis states. Here, the system is in an equal superposition over those states.

as its ground state is easy to compute. Measuring the system at this stage would collapse the superposition and yield one of the four states at random, each with equal probability.

During the annealing process, the Hamiltonian evolves and is given by:

$$\mathcal{H} = \left(1 - \frac{t}{T}\right) \sum_{i=0}^{N-1} \sigma_i^x + \frac{t}{T} \left(\sum_{i=0}^{N-1} h_i \sigma_i^z + \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} J_{i,j} \sigma_i^z \sigma_j^z \right), \tag{A.1}$$

where the coefficients on the initial and problem Hamiltonians correspond to the annealing schedule. Again, we use σ_z^i as a shorthand for $\sigma_i^z = I \otimes ... \otimes \sigma^z \otimes ... \otimes I$, where σ^z is the Pauli Z matrix:

$$\sigma^z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

The Hamiltonian (A.1) includes *transverse fields*, the σ_i^x terms, which pull qubits toward a superposition of the $+1$ and -1 states. These fields are in tension with the σ_i^z terms in the problem Hamiltonian, which pull qubits toward a classical computational basis state.[32]

A.1 Computational Complexity

Problems solved on a QA are mapped to a physical quantum Ising problem, which is NP-hard to solve exactly on a classical computer (Barahona, 1982). The associated *decision problem* is NP-complete, which means that it can be mapped to any other NP problem in a polynomial number of steps.

If a QA could solve the Ising problem exactly in polynomial time, it could, in principle, solve any NP problem in polynomial time. But both theory and experiments suggest this is unlikely. Instead, QAs appear to find high-quality approximate solutions quickly, but further improvements take more time (Zintchenko et al., 2015; King et al., 2015).

This is because the transition between the initial and problem Hamiltonians must be adiabatic to keep the system in its ground state. Faster transitions can lead to a jump in the energy level from the ground state to the first excited state, which can prevent the system from terminating in a state consistent with the global minimum at the end of the annealing process. Still, even if a QA cannot guarantee that the process is adiabatic, the system may end in the ground state or close to it. In cases where the spectral gap remains large throughout the transition process, a QA may transition sufficiently slowly to ensure that this happens.

Importantly, the inability to guarantee polynomial time solutions for combinatorial optimization problems does not rule out the potential for quantum advantage. Consider the case

[32] The eigenvectors of σ^z are $[1, 0]$ and $[0, 1]$, corresponding to $+1$ and -1 in the classical Ising model, or 0 and 1 in the computational basis. The eigenvectors of σ^x are $[1/\sqrt{2}, 1/\sqrt{2}]$ and $[1/\sqrt{2}, -1/\sqrt{2}]$, which are equal superpositions of the classical Ising states.

where we have exponential and polynomial time algorithms for the same problem, but the computational resource requirements are sufficiently high for both that only small problem instances can be solved. In this case, the exponential time algorithm might be able to solve larger problems. Figure A.1 illustrates this idea: the exponential-time algorithm (circles) and the polynomial-time algorithm (squares) intersect with the dashed resource constraint at different points. Even if QA scales exponentially, it may still allow us to solve harder problems faster than we could classically.

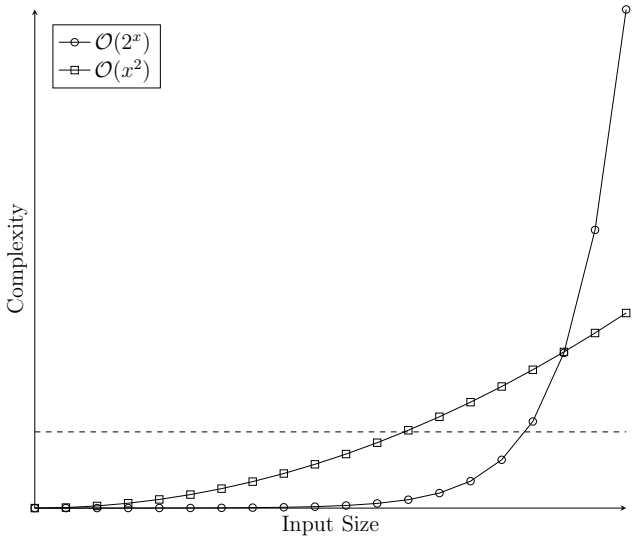


Figure A.1: Comparison of the computational complexity of algorithms with polynomial and exponential time complexity for different input sizes.

B Quadratization Methods

We discuss four quadratization methods that introduce either no auxiliary variables or the minimum number of auxiliary variables.

Deduction Reduction. Tanburn et al. (2015) propose a quadratization strategy that identifies properties that must hold in the ground state of the system. Once such properties are known, one can substitute terms in the pseudo-Boolean function to reduce its degree without introducing auxiliary variables. They illustrate this method with a simple constraint satisfaction

problem:

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ x_1 x_4 + x_2 x_5 &= x_3 \\ x_1 + 2x_2 &= x_3 + 2x_4. \end{aligned} \tag{B.2}$$

The problem can be expressed in PBF form as:

$$\begin{aligned} \mathcal{H} &= (x_1 + x_2 + x_3 - 1)^2 + (x_1 x_4 + x_2 x_5 - x_3)^2 + (x_1 + 2x_2 - x_3 - 2x_4)^2 \\ &= 2x_1 x_2 x_4 x_5 - 2x_1 x_3 x_5 - 2x_2 x_3 x_5 - 2x_2 x_3 + 6x_1 x_2 - 3x_1 x_4 - 8x_2 x_4 + \\ &+ x_2 x_5 + 3x_2 + 4x_3 x_4 + x_3 + 4x_4 + 1. \end{aligned} \tag{B.3}$$

For our purposes, it suffices to consider the first deduction-reduction. Equation (B.2) suggests that $x_1 x_2 = x_2 x_3 = x_3 x_1 = 0$ in the ground state. Naive substitution of this deduction into equation (B.3) reduces the Hamiltonian to the quadratic form:

$$\mathcal{H} = -3x_1 x_4 - 8x_2 x_4 + x_2 x_5 + 3x_2 + 4x_3 x_4 + x_3 + 4x_4 + 1.$$

Whereas the original Hamiltonian had a ground state energy level of $\mathcal{H} = 0$, the new Hamiltonian has states that yield $\mathcal{H} = -3$ and $\mathcal{H} = -2$. Thus, the substitution was not valid.

Tanburn et al. (2015) explain that this problem can be overcome by performing substitutions term-by-term and adding penalty terms where necessary. For instance, consider the quartic monomial term $2x_1 x_2 x_4 x_5$. It can either add 0 or 2 to the Hamiltonian. In the ground state, $x_1 x_2 = 0$, so it will add 0. However, it may add 2 or 0 outside of the ground state. Thus, imposing $x_1 x_2 = 0$ will lower the energy level for some non-ground states. Hence, we must add $2x_1 x_2$ as a penalty to $\mathcal{H}$ if we perform this substitution. That is, $2x_1 x_2 x_4 x_5 \to 2x_1 x_2$.

Excludable Local Configurations (ELCs). Ishikawa (2014) defines an ELC as a partial assignment of variables that makes it impossible to achieve the ground state (global minimum). ELCs can be used to perform degree reduction without adding auxiliary variables. Dattani (2019) provides an example of a Hamiltonian that contains an ELC:

$$\mathcal{H} = x_1 x_2 + x_2 x_3 + x_3 x_4 - 4x_1 x_2 x_3. \tag{B.4}$$

Since $x_i \in \{0, 1\}$, any state where $x_1 x_2 x_3 = 1$ will have lower energy than any state where $x_1 x_2 x_3 = 0$. This is because each of the first three terms on the right side of equation (B.4) will either be equal to 0 or 1, whereas, the fourth term will be equal to 0 or -4. Therefore, a partial assignment such as $(x_1, x_2, x_3) = (1, 0, 0)$ is excludable, since $x_1 x_2 x_3 = 0$, which cannot

be a ground state. We can apply the algorithm below, paraphrased from Ishikawa (2014), to reduce the cubic term:

1. If the coefficient on the higher-order term ζ is negative, find an ELC with a parity that matches the size of the partial assignment. If it is even, then find an ELC with the opposite parity.

2. For the ELC a, add the term to the Hamiltonian $\xi(x) = |\zeta| \prod_i \{a_i x_i + (1 - a_i)(1 - x_i)\}$.

In our case, $\zeta = -4$ and the partial assignment is $(x_1, x_2, x_3) = (1, 0, 0)$. This has an odd number of variables and an odd parity (number of 1s). Thus, condition 1 is satisfied. We can compute $\xi(x)$ as in:

$$\xi(x) = |-4|(1 \cdot x_1 + 0 \cdot (1 - x_1))(0 \cdot x_2 + 1 \cdot (1 - x_2))(0 \cdot x_3 + 1 \cdot (1 - x_3)).$$

Adding $\xi(x)$ to $\mathcal{H}$ yields $\mathcal{H}' = x_1x_2 + x_2x_3 + x_3x_4 + 4x_1 - 4x_1x_2 - 4x_1x_3$, which has the same ground state as $\mathcal{H}$, but without the higher-order terms.

Negative Term Reduction (NTR). When a higher-order term is negative, we can use the NTR approach (Kolmogorov and Zabin, 2004; Freedman and Drineas, 2005) to reduce it to a sum of quadratic terms using only one auxiliary variable, x_a:

$$-\prod_{i=1}^{d} x_i \to (d-1)x_a - \sum_{i=1}^{d} x_i x_a.$$

This form reproduces the full energy spectrum, including the ground state. It can also reduce d-order terms to quadratic terms using a single auxiliary variable.

Positive Term Reduction (PTR). For positive monomial terms, no algorithms can reduce d-order terms to quadratics using only one auxiliary variable. However, Boros and Gruber (2014) demonstrate how to reduce a d-order term to a quadratic using $d-2$ auxiliary variables:

$$\prod_{i=1}^{d} x_i \to \left(\sum_{i=1}^{d-2} x_{a_i}(d - i - 1 + x_i - \sum_{j=i+1}^{d} x_j) \right) + x_{d-1}x_d.$$

C PPI Convergence Time and Errors

Figure C.2 plots the average results from 10 executions of Algorithm 1 on a classical computer. The horizontal axis shows the total execution time in microseconds, with each tick corresponding

to an iteration. The vertical axis displays the parameter errors as absolute percentage deviations from their true values (which, in our version of the RBC model, can be computed analytically).

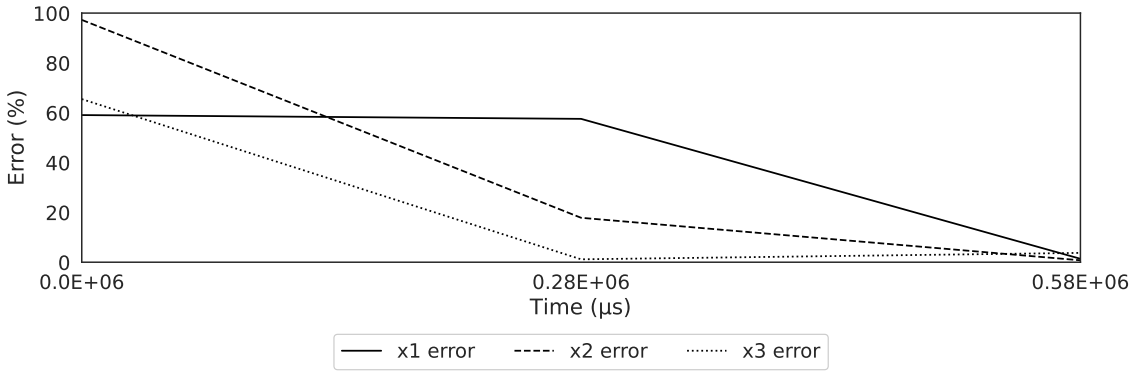


Figure C.2: Policy and value function parameter errors as a function of average execution time over 10 runs. Convergence is achieved after two iterations.

D Annealing Schedule Simulations

We consider two trivial problems that require an iterative solution. The first is encoded in the Hamiltonian

$$\mathcal{H}_1 = z_0 - z_1 - 2z_0z_1.$$

We set the initial state to $z_0 = z_1 = 0$ and apply Algorithm 5 with $\mathcal{C} = 1$. If z_0 is annealed first, it will be optimal to leave its value unchanged at 0, yielding $\mathcal{H}_1 = 0$. If z_1 is annealed next with $z_0 = 0$, it will be optimal to flip z_1 to 1, lowering $\mathcal{H}_1$ to -1. In the absence of noise, an annealer would return this as the candidate solution, even though the global minimum is $\mathcal{H}_1 = -3$ at $z_0 = z_1 = 1$. We must perform another iteration ($\mathcal{C} = 2$) to reach the global minimum.

The second problem, which is more challenging, is encoded in the Hamiltonian

$$\mathcal{H}_2 = z_2(2 + z_0 - 2z_0z_1) + z_3(2 - z_1 - 2z_0z_1),$$

and activates or deactivates components of the loss function using auxiliary qubits, z_2 and z_3. This provides additional control over the implementation of cycles and aligns more closely with Algorithm 5. As with the previous problem, initializing in state $z_0 = z_1 = z_2 = z_3 = 0$ ensures that multiple cycles are needed to reliably obtain the global minimum.

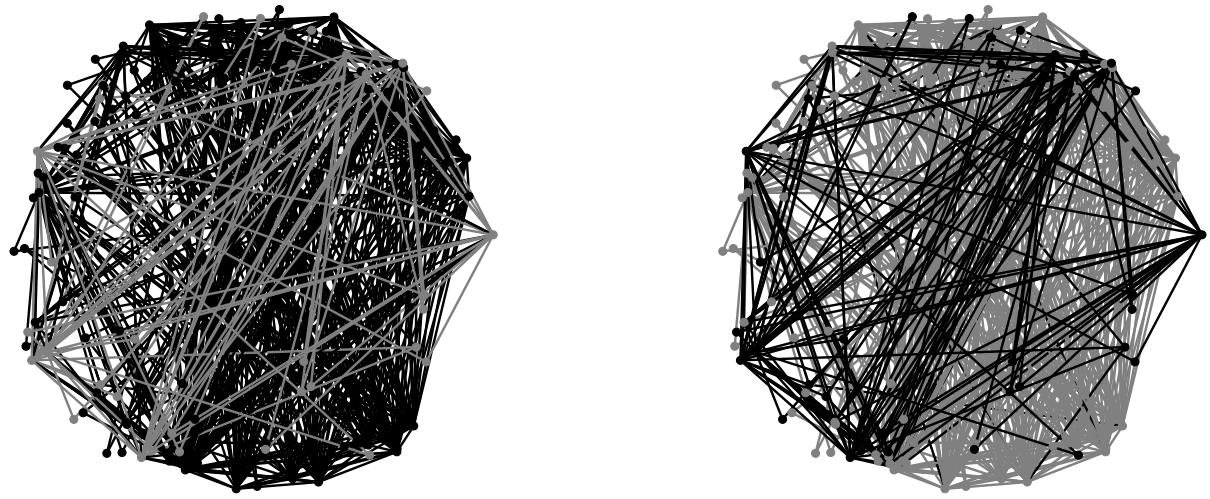
We show this experimentally in Table D.1, where $\mathcal{C} = 1$ yields the correct solution 0% of the time, but $\mathcal{C} = 2$ increases the success rate to 100%.

Cycles	Share Correct ($\mathcal{H}_1$)	Share Correct ($\mathcal{H}_2$)
1	0.10	0.00
2	0.80	1.00

Table D.1: Share of anneals that yield the correct solutions in the problems $\mathcal{H}_1$ and $\mathcal{H}_2$ given how many times the group 1 and group 2 annealing subschedules are repeated within a single anneal.

E Annealing Activity

The annealing activity in each step of Algorithm 4 is depicted as a weighted graph in Figure E.3. A gray node indicates a qubit in a classical state, while a black node indicates it is in a superposition state undergoing annealing. Black edges signify that at least one of the connected nodes is active in the anneal. Notice that the policy improvement step exhibits more annealing activity, as the quadratic terms used to approximate $\ln(x_1)$ require higher connectivity among qubits.



Policy Improvement Graph

Policy Valuation Graph

Figure E.3: State of the problem graph during the policy improvement and policy valuation steps.