PLOS ONE

RESEARCH ARTICLE

Controlled precision QUBO-based algorithm to compute eigenvectors of symmetric matrices

Benjamin Krakoff[1], **Susan M. Mniszewski**[2], **Christian F. A. Negre**[3]*

1 Department of Mathematics, University of Michigan, Ann Arbor, Michigan, United States of America, **2** Computer, Computational, and Statistical Sciences Division, Los Alamos National Laboratory, Los Alamos, New Mexico, United States of America, **3** Theoretical Division, Los Alamos National Laboratory, Los Alamos, New Mexico, United States of America

* cnegre@lanl.gov

Citation: Krakoff B, Mniszewski SM, Negre CFA (2022) Controlled precision QUBO-based algorithm to compute eigenvectors of symmetric matrices. PLoS ONE 17(5): e0267954. https://doi.org/10.1371/journal.pone.0267954

Editor: Itay Hen, University of Southern California, UNITED STATES

Received: July 22, 2021

Accepted: April 19, 2022

Published: May 9, 2022

Data Availability Statement: The breasttissue_10NN, mesh1em1, and spaceShuttleEntry 1 were downloaded from: https://sparse.tamu.ed Marchenko-Pastur matrices were generated following the method in: "Marčenko, V. A., and L. A. Pastur. 1967. "DISTRIBUTION OF EIGENVALUES FOR SOME SETS OF RANDOM MATRICES." Mathematics of the USSR-Sbornik. https://doi.org/10.1070/sm1967v001n04abeh001994.

Funding: The study was supported by National Science Foundation. The funders had no role in

Abstract

We describe an algorithm to compute the extremal eigenvalues and corresponding eigenvectors of a symmetric matrix which is based on solving a sequence of Quadratic Binary Optimization problems. This algorithm is robust across many different classes of symmetric matrices; It can compute the eigenvector/eigenvalue pair to essentially any arbitrary precision, and with minor modifications, can also solve the generalized eigenvalue problem. Performance is analyzed on small random matrices and selected larger matrices from practical applications.

1 Introduction

The problem of computing eigenvectors and eigenvalues to a desired precision has many applications in science and mathematics, including web page ranking [1], planar embeddings [2], and principal component analysis [3], among many others. The recent development of new computing paradigms has led to the development of *annealing devices*, which are specialized hardware designed to solve Quadratic Binary Optimization Problems (QUBOs). Such annealers include D-Wave's quantum annealers and Fujitsu's Digital Annealer. This has lead to a corresponding interest in reformulating computational tasks as QUBOs and solving them using these annealing devices. This strategy has been applied to several problems including graph partitioning [4], solving polynomial equations [5], and vertex coloring [6]. Here we compute eigenvectors of symmetric matrices by solving a sequence of QUBOs, which allow the eigenvectors and eigenvalues to be found to any desired precision. A mathematically similar approach to this problem is considered in [7, 8], but accuracy is increased by increasing the size of the associated QUBO. In contrast, the proposed algorithm can compute eigenvectors to essentially arbitrary precision without increasing the size of the QUBOs, which can have as few as twice as many variables as the original eigenvalue problem. The trade-off for using small QUBOs is that more iterations are required. A similar approach is considered in Appendix C of [9], although here the effects of different parameters are more thoroughly studied, and the

study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

presentation gives a very general optimization framework. The performance data is collected using D-Wave's Ocean Simulated Annealing (SA) package.

As mentioned, the state-of-art of QUBO eigensolver is proposed in [8]. In the aforementioned work, the only way of increasing precision is to increase the QUBO size by increasing the number of bits required to represent the real numbers. In the hereby proposed method we can compute the eigenpair to arbitrary any precision without increasing the size of the QUBO.

The paper is organized as follows. The relevant mathematical background for symmetric matrices and use of QUBO solvers as a descent method is explained in Section 2.1. The algorithm for computing the eigenvector/eigenvalue pair is given in Section 2.3. Experimental results with various parameters and matrices are presented in Section 3, followed by the conclusion in Section 4.

2 Methods

2.1 Mathematical background

Let A be a symmetric matrix. A well-known consequence of the spectral theorem is that the smallest eigenvalue λ and corresponding eigenvector $\mathbf{v}$ are global minima for the Rayleigh quotient $\mathbf{x}^tA\mathbf{x}/\mathbf{x}^t\mathbf{x}$

$$\lambda = \min_{\|\mathbf{x}\|=1} \mathbf{x}^tA\mathbf{x}, \quad \mathbf{v} = \operatorname*{argmin}_{\|\mathbf{x}\|=1} \mathbf{x}^tA\mathbf{x} \tag{1}$$

The proposed algorithm uses a QUBO formulation of the problem to both obtain a good initial guess for the global minimum, and to implement an iterative descent from the initial guess. Similar to classical descent methods such as Newton Conjugate-Gradient and the BFGS algorithms [10], this algorithm requires computing, but not inverting, a Hessian matrix at each descent step. We begin with an overview of QUBOs and how they can be used to approximately solve certain constrained quadratic optimization problems.

Let $\{0, 1\}^m$ denote the set of binary vectors of length m, and let Q be a symmetric $m \times m$ matrix. The combinatorial optimization problem

$$\operatorname*{argmin}_{\mathbf{x}_b\in\{0,1\}^m} \mathbf{x}_b^tQ\mathbf{x}_b$$

is called a *quantum unconstrained binary optimization* problem, or QUBO, and it is known to be NP-hard [11]. As mentioned before, interest in casting various problems as QUBOs has increased due to the development of annealing devices, which are a class of hardware that use ideas from statistical mechanics to produce approximate solutions to a QUBO. See for example [12] or [13].

To solve a real-variable optimization problem using a QUBO, we require a method of approximating each real variable by b binary variables. This number b will be a parameter referred to as the number of bits.

Let's start with a few concrete examples of the arithmetic involved, beginning with a demonstration of how to multiply two real numbers such as $x = -.5$ and $y = .5$ using 2 bits. Form the *precision vector* $\mathbf{p} = (-1, .5)$ with corresponding *precision matrix* $P = \mathbf{p}^t\mathbf{p} = \begin{pmatrix} 1 & -.5 \\ -.5 & .25 \end{pmatrix}$. Set $\mathbf{x}_b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $\mathbf{y}_b = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ so that

$$x = \mathbf{p} \cdot \mathbf{x}_b, \quad y = \mathbf{p} \cdot \mathbf{y}_b \tag{2}$$

$$-.25 = x \cdot y = \mathbf{x}_b^t \mathbf{p}^t \mathbf{p} \mathbf{y}_b = (1,1)\begin{pmatrix} 1 & -.5 \\ -.5 & .25 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{3}$$

Now let Q be a symmetric matrix, and we shall demonstrate how to compute the quadratic form $(x, y, z)Q\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ using binary variables. As above, set $z = \mathbf{p} \cdot \mathbf{z}_b$ where $\mathbf{z}_b$ is a binary vector and z is the corresponding real number. We can rewrite the quadratic form as

$$(\mathbf{x}_b^t, \mathbf{y}_b^t, \mathbf{z}_b^t)\begin{pmatrix} \mathbf{p}^t & & \\ & \mathbf{p}^t & \\ & & \mathbf{p}^t \end{pmatrix} Q \begin{pmatrix} \mathbf{p} & & \\ & \mathbf{p} & \\ & & \mathbf{p} \end{pmatrix}\begin{pmatrix} \mathbf{x}_b \\ \mathbf{y}_b \\ \mathbf{z}_b \end{pmatrix} \tag{4}$$

The middle three terms can be written more succinctly as $(I_3 \otimes p^t)Q(I_3 \otimes p) = Q \otimes P$ where I_3 is the 3×3 identity matrix and $\otimes$ is the tensor product.

Now we describe the construction in full generality. Given a *precision vector* $\mathbf{p} = \left(-1, \frac{1}{2}, \frac{1}{2^2}, \frac{1}{2^3}, \ldots \frac{1}{2^{b-1}}\right)$ of length b, the set of integer multiples of $\frac{1}{2^{b-1}}$ in the interval $\left[-1, 1 - \frac{1}{2^b}\right]$ is exactly the set

$$C_{1,b} := \{\mathbf{p} \cdot \mathbf{x}_b | \mathbf{x}_b \in \{0,1\}^b\} \tag{5}$$

We use the sub-scripted $\mathbf{x}_b$ as a convention to emphasize that $\mathbf{x}_b$ is a binary vector, i.e. the sub-script does not refer to the number of bits. More generally, the n-fold product of $C_{1,b}$ is the set

$$C_{n,b} := \{(I_n \otimes \mathbf{p})\mathbf{x}_b | \mathbf{x}_b \in \{0,1\}^{nb}\} \tag{6}$$

where I_n is the identity matrix. The set $C_{n,b}$ will be referred to as a discretized cube. Let Q be a symmetric $n \times n$ matrix, $\mathbf{r}$ an n-vector and suppose we want to solve the constrained quadratic programming problem

$$\underset{\mathbf{x} \in [-1,1]^n}{\text{argmin}}\ \mathbf{r}^t\mathbf{x} + \mathbf{x}^t Q \mathbf{x} \tag{7}$$

To get an approximate solution to (7) using a QUBO, first replace the unit cube by a discretized unit cube to get the optimization problem.

$$\underset{\mathbf{x} \in C_{n,b}}{\text{argmin}}\ \mathbf{r}^t\mathbf{x} + \mathbf{x}^t Q \mathbf{x} \tag{8}$$

Setting $m = nb$, $\mathbf{x} = (I_n \otimes \mathbf{p})\mathbf{x}_b$, and $P = \mathbf{p}^t\mathbf{p}$, this is equivalent to the following QUBO problem:

$$\underset{\mathbf{x}_b \in \{0,1\}^m}{\text{argmin}}\ \mathbf{r}^t(I_n \otimes \mathbf{p})\mathbf{x}_b \quad + \mathbf{x}_b^t(Q \otimes P)\mathbf{x}_b \tag{9}$$

$$= \underset{\mathbf{x}_b \in \{0,1\}^m}{\text{argmin}}\ \mathbf{x}_b^t \text{Diag}(\mathbf{r}^t I_n \otimes \mathbf{p})\mathbf{x}_b + \mathbf{x}_b^t(Q \otimes P)\mathbf{x}_b \tag{10}$$

$$= \underset{\mathbf{x}_b \in \{0,1\}^m}{\text{argmin}}\ \mathbf{x}_b^t(\text{Diag}(\mathbf{r}^t I_n \otimes \mathbf{p}) + Q \otimes P)\mathbf{x}_b \tag{11}$$

Here Diag($\mathbf{v}$) refers to the diagonal matrix with entries from $\mathbf{v}$. Going from lines (9) to (10) uses the following identity valid for binary vectors: $\mathbf{v}^t\mathbf{x}_b = \mathbf{x}_b^t\text{Diag}(\mathbf{v})\mathbf{x}_b$.

To summarize, given a number of bits b, this procedure approximates the real optimization problem in n variables (7) with the QUBO (11) of size $n \cdot b$. We conclude by remarking that we are not restricted to the cube $[-1, 1]^n$. If we instead want to optimize over the cube $[-\delta, \delta]^n$, we need to repeat the same construction with the precision vector $\delta \cdot \mathbf{p}$. An immediate question is how much error is introduced by replacing the cube with a discretized cube.

Lemma 1. *Lipschitz Estimate*

Let $\mathbf{x}_T$ *be an optimal solution to* (7), *and let* $\mathbf{x}_A$ *be an optimal solution to* (8). *Then*

$$|\mathbf{r}^t\mathbf{x}_A + \mathbf{x}_A^t Q\mathbf{x}_A - (\mathbf{r}^t\mathbf{x}_T + \mathbf{x}_T^t Q\mathbf{x}_T)| \leq \sqrt{\frac{n}{2^b}} \sup_{[-1,1]^n} \|2Q\mathbf{x} + \mathbf{r}\| \tag{12}$$

Proof. Let $\hat{\mathbf{x}_T}$ be the point in $C_{n,b}$ closest to $\mathbf{x}_T$. The gradient of $\mathbf{r}^t\mathbf{x} + \mathbf{x}^t Q\,\mathbf{x}$ is $2Q\mathbf{x} + \mathbf{r}$, and $\|\hat{\mathbf{x}_T} - \mathbf{x}_T\| \leq \sqrt{\frac{n}{2^b}}$, leading to the Lipschitz estimate

$$|\mathbf{r}^t\hat{\mathbf{x}_T} + \hat{\mathbf{x}_T}^t Q\hat{\mathbf{x}_T} - (\mathbf{r}^t\mathbf{x}_T + \mathbf{x}_T^t Q\mathbf{x}_T)| \leq \sqrt{\frac{n}{2^b}} \sup_{[-1,1]^n} \|2Q\mathbf{x} + \mathbf{r}\| \tag{13}$$

Combining (13) with the inequality $\mathbf{r}^t\mathbf{x}_T + \mathbf{x}_T^t Q\mathbf{x}_T \leq \mathbf{r}^t\mathbf{x}_A + \mathbf{x}_A^t Q\mathbf{x}_A \leq \mathbf{r}^t\hat{\mathbf{x}_T} + \hat{\mathbf{x}_T}^t Q\hat{\mathbf{x}_T}$ we get the following implication:

$$|\mathbf{r}^t\mathbf{x}_A + \mathbf{x}_A^t Q\mathbf{x}_A - (\mathbf{r}^t\mathbf{x}_T + \mathbf{x}_T^t Q\mathbf{x}_T)| \leq \sqrt{\frac{n}{2^b}} \sup_{[-1,1]^n} \|2Q\mathbf{x} + \mathbf{r}\| \tag{14}$$

Lemma 1 makes a trade-off apparent. With more bits, the solution on the discretized cube will better approximate the true solution of (7) but will require solving a larger QUBO.

Indeed, numerical experiments from subsequent sections will show that $b = 2$ generally requires more iterations than $b = 8$, indicating that the quality of the approximate solution at each step is worse, although interestingly using $b = 2$ takes less time overall since solving smaller QUBOs requires less computational effort. It is also worth noting that estimate (12) does not control the actual distance between solutions, $\|\mathbf{x}_A - \mathbf{x}_T\|$. In the special case when Q is positive definite, this distance can be controlled, but it would be interesting to have estimates in greater generality.

The annealers that one works with in practice are never ideal, and so will rarely return the absolute best solution $\mathbf{x}_A$ but instead a response consisting of many samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l$ of good solutions with energies $E(\mathbf{x}_i) \leq E(\mathbf{x}_{i+1})$. (Here *energy* of a solution $\mathbf{x}_i$ refers to the value of the objective function at $\mathbf{x}_i$). An obvious approach is to treat the lowest energy solution $\mathbf{x}_0$ as the best approximation of $\mathbf{x}_T$. A subtler approach that can reap great benefits in practice is to take a linear combination of the full response:

$$\frac{1}{l}\sum_{i=1}^{l} e^{-\beta(E(\mathbf{x}_i)-E(\mathbf{x}_0))}\mathbf{x}_i \tag{15}$$

as an approximation of $\mathbf{x}_T$, where β is a parameter. Experiments in later sections were conducted either using the best response $\mathbf{x}_0$ or the full response with $\beta = 100$ and performance is compared for several values of n and b. Although (15) is an *ad hoc* method, the reason for why it works could be due to the energy distribution of the QUBO results. Provided the probability of getting an exited state is Boltzmann distributed (Fermi distribution limit for a single particle at low temperature limits); some responses with considerable low energies could introduce "bit flips" that might correct for the discretization error when added as a linear combination with weights that would decay as an exponential of their energies. A more formal explanation using quantum dynamics for why equation 14 works deserves a further study.

Another approach to get better approximations of $\mathbf{x}_T$ is to solve a sequence of QUBOs with bias. More precisely, get an initial approximation of $\mathbf{x}_T$ by following the previous procedure to produce $\mathbf{x}_{T_1}$. Then modify the QUBO by adding a linear term $-\alpha \mathbf{x}_{T_1}^t \mathbf{x}$ where $\alpha > 0$ and find approximate solutions to

$$\underset{C_{n,b}}{\operatorname{argmin}} \quad \mathbf{r}^t\mathbf{x} + \mathbf{x}^t Q\mathbf{x} - \alpha \mathbf{x}_{T_1}^t \mathbf{x} = \underset{C_{n,b}}{\operatorname{argmin}} \quad (\mathbf{r} - \alpha \mathbf{x}_{T_1})^t \mathbf{x} + \mathbf{x}^t Q\mathbf{x} \tag{16}$$

By Cauchy-Schwartz, $\frac{\mathbf{x}_{T_1}}{\|\mathbf{x}_{T_1}\|} = \underset{\|\mathbf{x}\|=1}{\operatorname{argmin}} \quad -\mathbf{x}_{T_1}^t \mathbf{x}$, thus in solving (16) the annealer is encouraged to produce solutions in the direction of $\mathbf{x}_{T_1}$. The annealer produces a new lower-energy solution $\mathbf{x}_{T_2}$ and this process can be repeated until the new solution no longer has lower energy than the previous. Experimental results in later sections contain data with $\alpha = 0$ and.1. Biasing is most helpful in the initial phase of the algorithm when it is iteratively producing solutions close to previous solutions. In later phases biasing is less useful, as will be evident from the results.

2.2 An iterative descent algorithm

Algorithms that solve continuous optimization problems rely on a good initial guess and an iterative descent rule. These tasks can be formulated as QUBO problems when trying to minimize the Rayleigh quotient over the unit sphere.

2.2.1 Obtaining an initial guess. Let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be the eigenvalues of A. To get an initial approximation of λ_1, one can ask to solve

$$\underset{\mathbf{x}\in C_{n,b}}{\operatorname{argmin}} \ \mathbf{x}^t A\mathbf{x} \tag{17}$$

as an approximation of

$$\underset{\|\mathbf{x}\|=1}{\operatorname{argmin}} \ \mathbf{x}^t A\mathbf{x} \tag{18}$$

An immediate problem is that if A is positive definite, the solution to (17) is just $\mathbf{x} = 0$. This can be remedied by replacing A with $A - \lambda I_n$, where $\lambda \in (\lambda_i, \lambda_{i+1})$ for some i. The eigenvectors are unaffected, the eigenvalues can be recovered from the new matrix and the solutions to

$$\underset{\mathbf{x}\in C_{n,b}}{\operatorname{argmin}} \ \mathbf{x}^t(A - \lambda I_n)\mathbf{x} \tag{19}$$

tend to be long, nonzero vectors very close to the span of eigenvectors which have negative eigenvalues for $A - \lambda I_n$, namely $\mathbf{v}_1, \ldots, \mathbf{v}_i$. A good initial choice is the average of the eigenvalues $\lambda = \frac{tr(A)}{n}$. This is a natural choice since it ensures the initial guess to be within the bounds of the eigenspectrum. As the algorithm progresses, λ will decrease towards λ_1. To converge in fewer iterations, it's better to choose λ close to, but greater than λ_1, as we will examine later.

These observations lead to the following iterative fixed-point method to produce a good initial guess for the lowest eigenvector. Initially solve (19) with $\lambda = \frac{tr(A)}{n}$ to produce a guess $\mathbf{v}_1$. Update λ using the Rayleigh quotient $\lambda = \frac{\mathbf{v}_1^t A \mathbf{v}_1}{\|\mathbf{v}_1\|^2}$ and solve (19) again possibly using $\mathbf{v}_1$ as a bias vector to produce a second guess $\mathbf{v}_2$. Repeat until λ is no longer decreasing.

For small matrices, say 10×10, this procedure often suffices to produce the lowest eigenvalue with 2-3 digits of accuracy and the corresponding eigenvector to within a distance of order.1 of the true eigenvector. The descent stage of the algorithm increases the precision to essentially arbitrary order as it will be explained in the next section.

2.2.2 Iterative descent. Suppose we want to minimize a function $f : \mathbb{R}^m \to \mathbb{R}$, and let ∇f and $H(f)$ denote the gradient and Hessian of f, respectively. Starting with an initial guess $\mathbf{x}_0$, a common strategy is to Taylor expand f around $\mathbf{x}_0$

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f \cdot \delta + \delta^t \frac{H(f)}{2} \delta + o(\|\delta\|^3) \tag{20}$$

$$\delta := \mathbf{x} - \mathbf{x}_0 \tag{21}$$

and choose δ to minimize $\nabla f \cdot \delta$, which is gradient descent, or to minimize $\nabla f \cdot \delta + \delta^t \frac{H(f)}{2} \delta$, which includes second order methods such as Newton's method, BFGS, Newton Conjugate-Gradient, etc. Once a better solution $\mathbf{x}_1 = \mathbf{x}_0 + \delta$ has been found, Taylor expand around $\mathbf{x}_1$ again and repeat. The proposed algorithm obtains a good descent direction by using a QUBO to find good approximate solutions to

$$\underset{\delta \in C_{n,b}}{\operatorname{argmin}} \ \nabla f \cdot \delta + \delta^t \frac{H(f)}{2} \delta$$

Similar to Newton-CG and BFGS, this method requires computing, but not inverting, the Hessian matrix, and benefits from a line search which possibly increases the size of δ. See [10] for more details on classical optimization algorithms and the benefits of line search. Here the line search step amounts to minimizing a quadratic expression, and so the optimal scaling can be directly computed.

If k^{th} approximate solution $\mathbf{x}_k$ is closer to the true solution than any point in the discretized cube, one cannot expect minimizing the QUBO to produce a better solution. A key part of the descent phase is enforcing a minimum step size in addition to the line search so that the candidate $k + 1^{st}$ solution is possibly *worse* than that k^{th}. If the candidate solution is worse, the algorithm discards the candidate and replaces the discretized unit cube $C_{n,b}$ by a scaled-down discretized cube $t \cdot C_{n,b}$ where $t \ll 1$, which amounts to repeating the procedure outlined in section 3 with the precision vector $t \cdot \mathbf{p}$. Once the discretized cube has been scaled down, the algorithm continues running until it needs to scale down the cube further, or exits having achieved the desired accuracy.

2.3 The algorithm

With the key ingredients covered, we are in a position to present the algorithm. As a reminder, at each step, the objective function is of the form $f(\mathbf{x}) = \mathbf{x}^t(A - \lambda I_n)\mathbf{x}$, whose gradient and Hessian can be calculated as $\nabla f = 2(A - \lambda I_n)\mathbf{x}^t$ and $H(f) = 2(A - \lambda I_n)$ respectively. These formulas are implicitly used in the descent phase of the algorithm. At several stages, the algorithm solves optimization problems of the form $\underset{\mathbf{x} \in t \cdot C_{n,b}}{\operatorname{argmin}} \ \mathbf{r}^t\mathbf{x} + \mathbf{x}^t A\mathbf{x}$. These are turned in to QUBOs as explained in section 2.1, and annealers are used to minimize the QUBOs that appear, possibly using full responses or biasing. In subsequent section the effects of biasing, full responses and other parameters will be analyzed.

Algorithm 1 Controlled Precision QUBO-based Algorithm to Compute Eigenvectors of Symmetric Matrices

```
Inputs: Symmetric n × n matrix A, bits for precision vector b,
desired precision ε_tol
Outputs: Smallest evec v and eval λ within ε_tol of true values
λ ← tr(A)/n
H ← A − λI_n                    // Enforcing Indefiniteness
v ← argmin_{C_{n,b}} x^t H x    // Initial Guess Phase
```

$\mathbf{v} \leftarrow \frac{\mathbf{v}}{\|\mathbf{v}\|}$
while $\mathbf{v}^t A\mathbf{v} < \lambda$ **do**
  $\lambda \leftarrow \frac{\mathbf{v}^t A\mathbf{v}}{\|\mathbf{v}\|^2}$
  $H \leftarrow A - \lambda \cdot I_n$
  $\mathbf{v} \leftarrow \underset{C_{n,b}}{\text{argmin}}\ \mathbf{x}^t H\mathbf{x}$
  $\mathbf{v} \leftarrow \frac{\mathbf{v}}{\|\mathbf{v}\|}$
end while
$precision \leftarrow .1$ // Descent Phase
while $precision > \epsilon_{tol}$ **do**
  $H \leftarrow A - \lambda I_n$
  $\delta \leftarrow \underset{precision \cdot C_{n,b}}{\text{argmin}}\ 2\mathbf{v}^t H\delta + \delta H\delta$ // Getting Descent Direction
  $\delta \leftarrow \delta - \langle \mathbf{v}, \delta \rangle \delta$ // Orthogonalizing δ, $\mathbf{v}$
  $t_{min} = \max\{-\mathbf{v}^t H\delta/(\delta^t H\delta), 1\}$ // Line search step
  $\delta \leftarrow t_{min} \cdot \delta$
  if $\frac{(\mathbf{v}+\delta)^t A(\mathbf{v}+\delta)}{\|\mathbf{v}+\delta\|^2} < \lambda$ **then** // Checking if solution improves
    $\mathbf{v} \leftarrow \frac{\mathbf{v}+\delta}{\|\mathbf{v}+\delta\|}$
    $\lambda \leftarrow \mathbf{v}^t A\mathbf{v}$
  else
    $precision \leftarrow .1 \cdot precision$ // Increasing precision otherwise
  end if
end while
return $\mathbf{v}$, λ

Two steps merit a bit more explanation. Replacing δ by $\delta - \langle \mathbf{v}, \delta \rangle \delta$ forces $\mathbf{v}, \delta$ to be orthogonal. Since the Rayleigh quotient needs to be optimized over the sphere, the update direction δ should be tangent to the sphere at $\mathbf{v}$, and the tangent space of the sphere at $\mathbf{v}$ is precisely the set of vectors orthogonal to $\mathbf{v}$. Second, the scaling δ is computed as $t_{min} = \max\{-\mathbf{v}^t H\delta/(\delta^t H\delta), 1\}$. The expression $-\mathbf{v}^t H\delta/(\delta^t H\delta)$ is the line search step coming from minimizing the quadratic $(\mathbf{v} + t_{min}\delta)^t H(\mathbf{v} + t_{min}\delta)$. Strictly speaking this quadratic expression only has a minimum when $\delta^t H\delta$ is positive, and in practice when using this algorithm it almost always is, and if not, set $t_{min} = 1$. A minimum scaling $t_{min} \geq 1$ is enforced so that the candidate update $\mathbf{v} + t_{min}\delta$ possibly overshoots the exact solution, resulting in a worse estimate of the lowest eigenvector. Overshooting is an indication that the discretized cube is no longer fine enough to produce better solutions, and so the candidate update is discarded and the discretized cube is scaled down. Intuitively the scaling at each step should be about the order of $\frac{1}{2^{b-1}}$, and the numerical experiments below all use a.1 factor.

Oftentimes in practice one wishes to solve a generalized eigenvalue problem of the form $A\mathbf{v} = \lambda B\mathbf{v}$. In the case when A, B are symmetric and B is strictly positive definite, the smallest generalized eigenvalue minimizes the generalized Rayleigh quotient

$$\lambda = \min_{\|\mathbf{x}\|=1} \frac{\mathbf{x}^t A\mathbf{x}}{\mathbf{x}^t B\mathbf{x}} \quad \mathbf{v} = \underset{\|\mathbf{x}\|=1}{\text{argmin}} \frac{\mathbf{x}^t A\mathbf{x}}{\mathbf{x}^t B\mathbf{x}} \tag{22}$$

The following small changes solves the generalized eigenvalue problem, again to essentially arbitrary precision. First, instead of initializing λ as $\frac{tr(A)}{n}$, one can generate a random unit vector $\mathbf{w}$ (or use a specified vector) and initialize $\lambda = \frac{\mathbf{w}^t A\mathbf{w}}{\mathbf{w}^t B\mathbf{w}}$. Second, replace every Rayleigh quotient with the corresponding generalized Rayleigh quotient. Lastly, instead of updating H as $H = A - \lambda I_n$, update as $H = A - \lambda B$, as the latter preserves the B-eigenspectrum of A, while the former does not.

We conclude by emphasizing that this algorithm reaches arbitrary precision without increasing the size of the QUBOs, all of which involve $n \cdot b$ binary variables. Additionally, all

quadratic problems are of the form $\mathbf{r}^t\,\mathbf{x} + \mathbf{x}^t(A - \lambda I_n)\mathbf{x}$ where A is fixed, implying that the potential non-zero coefficients of the QUBO do not change (examine formula (11)).

3 Experimental results

The algorithm and its variants are tested on a class of random matrices of varying sizes and on benchmark sparse matrices using D-Wave's simulated annealing (SA) software unless otherwise specified. For each experiment, the algorithm ran until the Rayleigh quotient of the approximate eigenvector was within 10^{-8} of the true value. The *total annealing time* is hence computed as the total amount of time the algorithm spends on performing SA. The SA is the bottle-neck for the algorithm, and we include the time to give the reader a sense of how the run-time varies with the size of the matrix n and the number of bits b, as we will demonstrate in subsequent sections. Adjusting the time for each anneal will not affect the accuracy of the algorithm, as it can reach desired precision with any reasonably good annealer, but may affect the run-time. For a specific architecture, one might be able to tune the anneal time optimally based on n and b, which is not a question we attempt to answer here.

3.1 Basic performance

First, we demonstrate the convergence as a function of the number of iterations using example matrices from the TAMU SuiteSparse collection [14]. Fig 1 shows performance on the breast-tissue_10NN matrix, a weighted graph adjacency matrix of size 106 × 106 for 2, 4, 6 and 8 bits using best response and no biasing.

Interestingly using fewer bits gives less time to reach desired accuracy despite requiring more iterations. A *log* – *log* regression on the MP matrices (see next section) gives that the anneal time grows like $(n \cdot b)^{1.57}$ and the number of iterations grows like $n^{.44}b^{-.32}$ so the total time is roughly $n^2 b^{1.2}$. The algorithm works on even larger matrices, as is demonstrated in Fig 2 using the spaceShuttleEntry_1 matrix, a 560 × 560 control matrix.

Fig 3 demonstrates the performance for the generalized eigenvalue problems using mesh1em1 as the A matrix and meshe1 as the B matrix, two 48 × 48 matrices from the SuiteSparse database.

In order to try to get algorithms that run as fast as possible, one might ask if it is possible get the algorithm to work using 1 bit of precision. With the current scheme this cannot be done. However, by reformulating the problem as an Ising instead of QUBO, one can indeed use only one bit precision. Ising problems are of the form

$$\underset{\mathbf{x}\in\{-1,1\}^n}{\operatorname{argmin}}\ \mathbf{h}^t\mathbf{x} + \mathbf{x}^t Q\mathbf{x}$$

the main distinction from QUBOs being the spin variables ±1. QUBOs or Ising problems are mathematically equivalent, and most annealers are capable of solving either.

Using the Ising formulation, its possible to mimic the same algorithm, which works well on very small matrices. However, for larger matrices, such as for the 106 × 106 weighted adjacency matrix the single-bit version of the algorithm takes longer than using two bits, taking 32.3 seconds and requiring over 400 iterations (compare with Fig 1). An educated guess for why this might happen follows. Since the solutions produced by Ising problems have coordinates that are all non-zero and of the same magnitude, if the algorithm has already produced a solution whose k^{th} coordinate is close to the true value, the added solution from the Ising problem will force that coordinate away from the optimal value. Using two bits is effective because the solutions can have coordinates that are positive, negative, or zero.

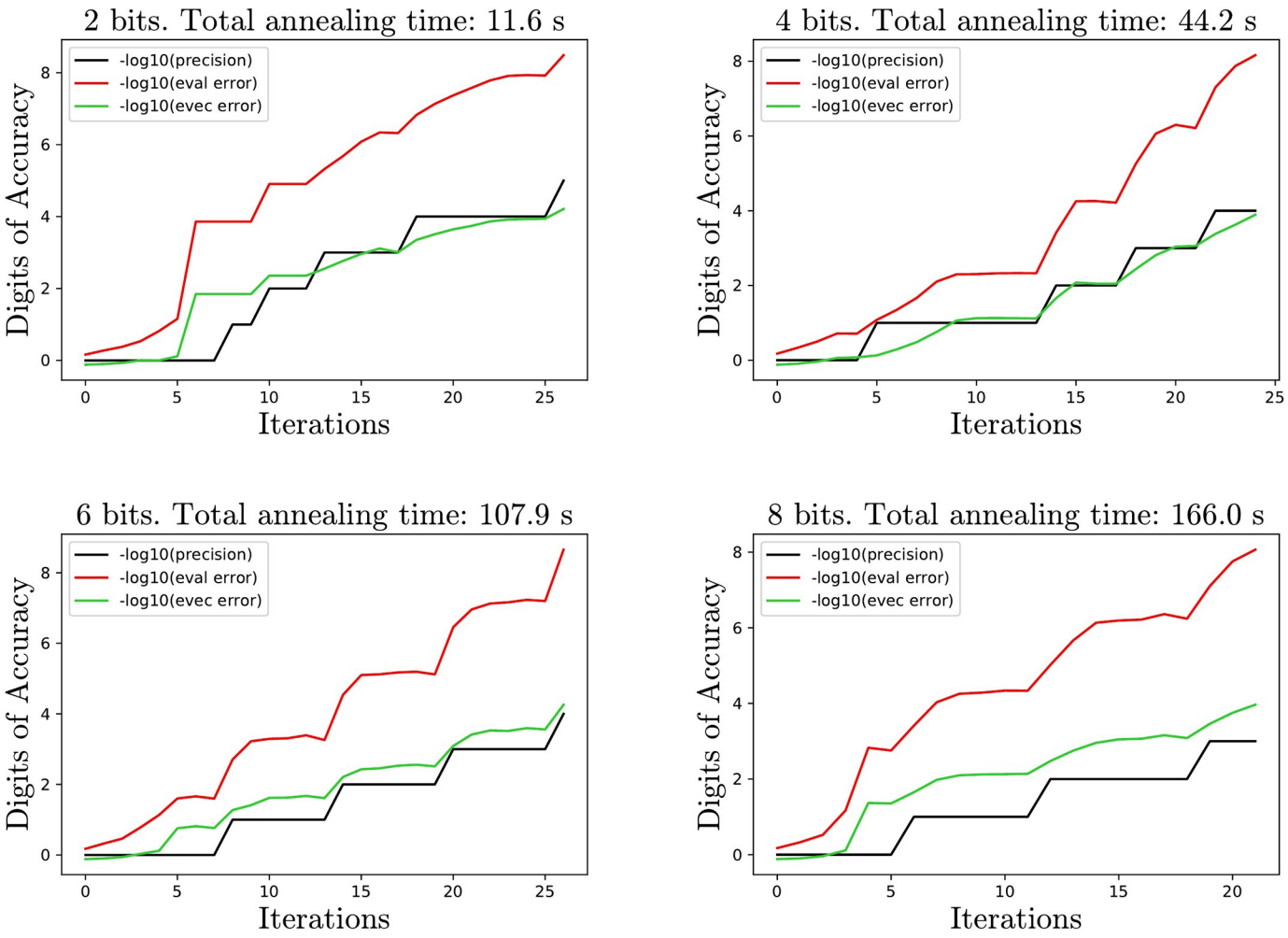

Fig 1. Digits of accuracy plotted against number of iterations. *Total annealing time* is the total amount of time the algorithm spends on performing SA, since SA dominates the cost of the method. *Evec error* is the distance of the computed vector from the true unit eigenvector. *Precision* refers to the scaling applied to the discretized cube at each iteration. The initial guess phase corresponds to a precision of 1. Observe that whenever the error increases, the algorithm responds by increasing the precision, which often gives large accuracy gains within the subsequent 2-3 iterations.

https://doi.org/10.1371/journal.pone.0267954.g001

3.2 Analysis of the parameters

To demonstrate the effect of biasing and full response parameters, the algorithm is tested on small matrices of sizes 3, 10 and 20 with number of bits $b = 2, 4, 6$ and 8. In the interest of not overwhelming the reader with plots and data, only the data for matrices of size 10 and 20 is displayed. We analyze the error at the end of the initial guess phase, and the average number of iterations each method requires. For each choice of size, bits and parameters, 10 Marchenko-Pasture matrices [15] with parameter $\lambda = .3$ are generated and the average errors at the end of the descent phase is recorded. Ideally this initial phase should end with the smallest possible error before beginning the descent phase. Towards this end, taking full responses (Eq (15)) and biases (Eq (16)) can be very beneficial. However, this benefit fades as the sizes of the QUBOs increase as one can see from Figs 4 and 5.

The choice to initialize λ as $\frac{tr(A)}{n}$ is motivated by a desire to produce an initial guess which is close to, but greater than, the true lowest eigenvector. To demonstrate this effect on 10×10 and 20×20 matrices, we compare performance initializing as $\frac{tr(A)}{n}$, which is the average of all

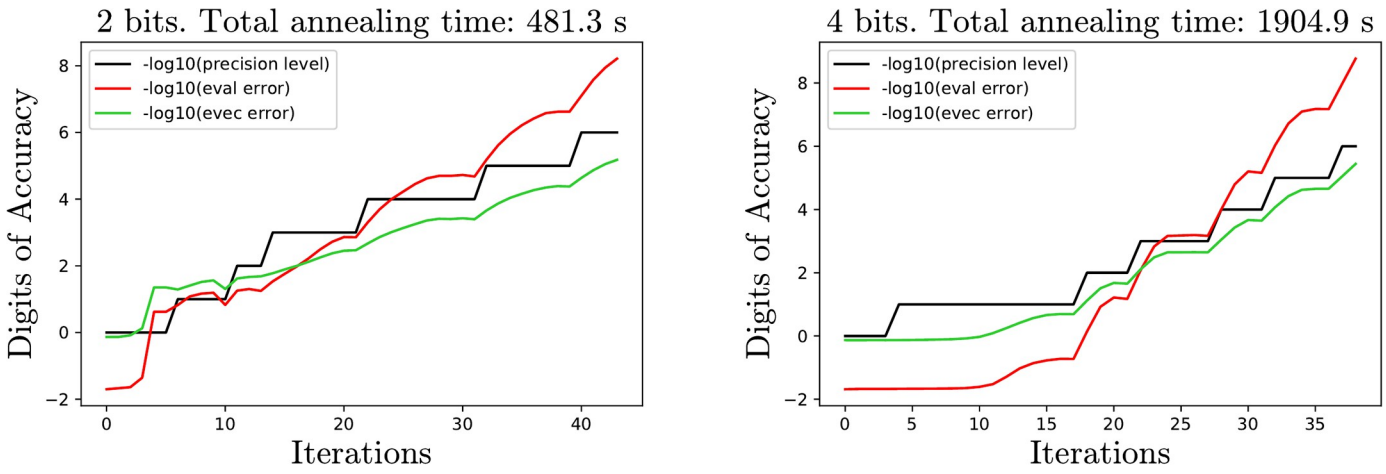

Fig 2. Error plot for 560 × 560 space Shuttle control matrix.

https://doi.org/10.1371/journal.pone.0267954.g002

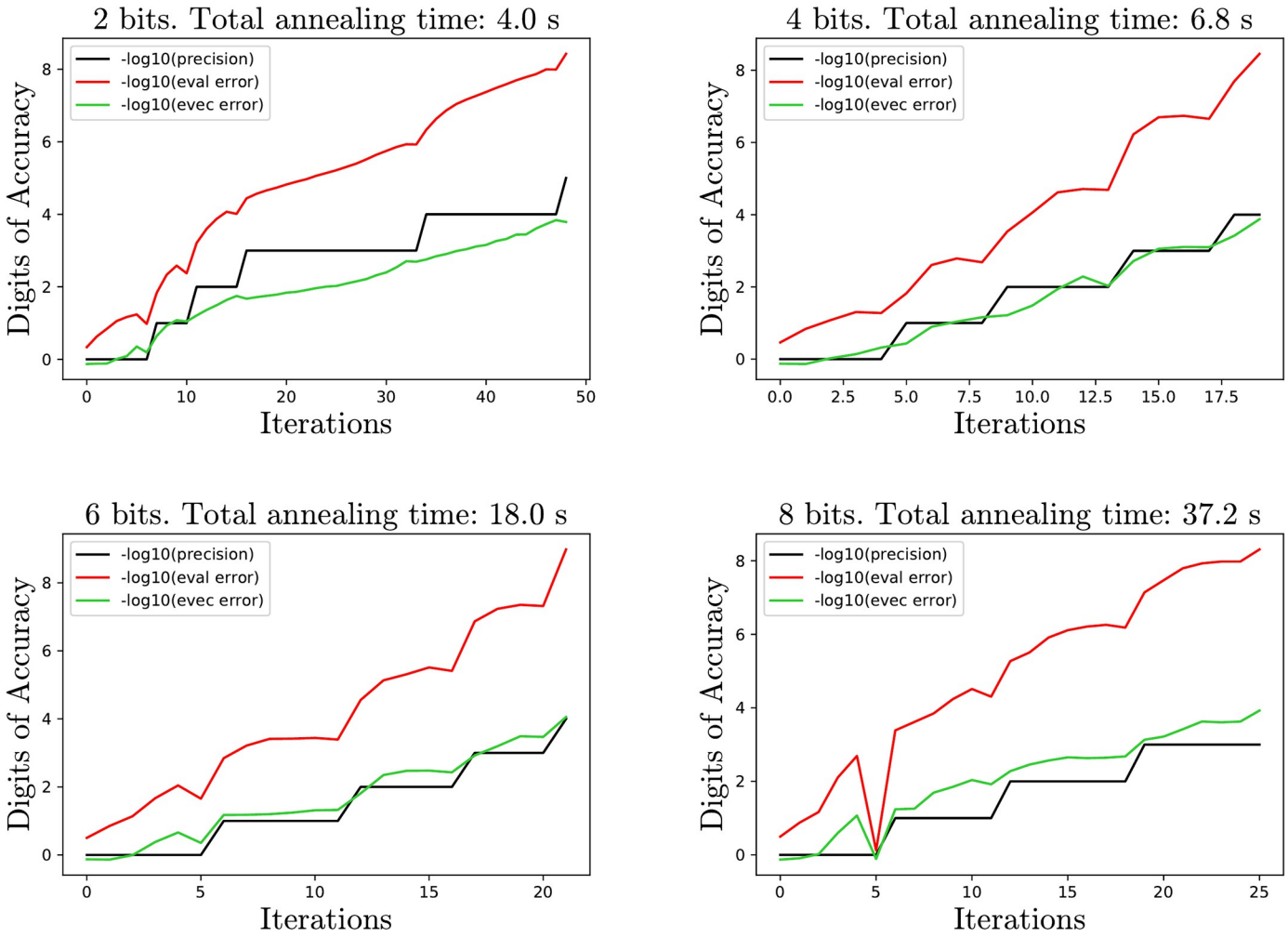

Fig 3. Error plots for generalized eigenvalue problem on two 48 × 48 mesh matrices.

https://doi.org/10.1371/journal.pone.0267954.g003

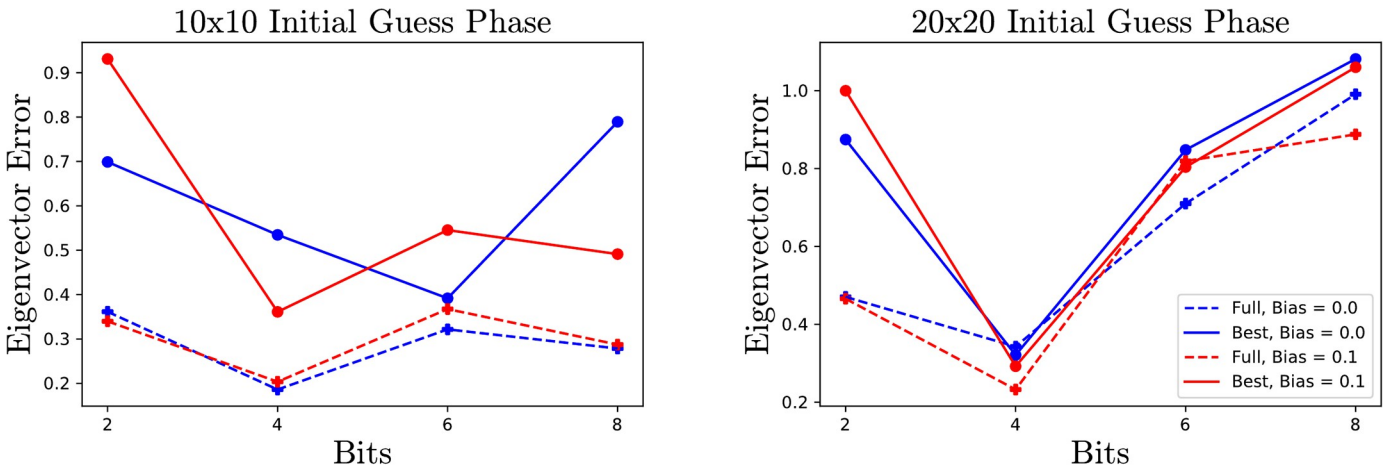

Fig 4. Eigenvector error for MP matrices at the end of initial guess phase. Observe that for small QUBOs the full response decreases the error significantly.

https://doi.org/10.1371/journal.pone.0267954.g004

the eigenvalues against initializing as the highest Gershgorin bound, which upper bounds the maximum eigenvalue [16]. Choosing an initialization closer to the true eigenvalue often leads to fewer iterations, although the difference is somewhat small and fades as the number of bits increases, as seen in Fig 6.

3.3 Gap size analysis

Here we analyze the effect of the spacing between eigenvalues. In particular, the gap $|\lambda_1 - \lambda_2|$ can significantly affect the number of iterations required to reach a given precision. For this experiment, given a gap size $g = |\lambda_1 - \lambda_2|$, an orthogonal matrix U is chosen at random with respect to the Haar measure using the SciPy implementation of [17]. The algorithm is then analyzed on the matrix $U^t\text{Diag}(0, g, 1, \ldots, n-2)U$. As Fig 7 demonstrates, as the gap size decreases the algorithm takes longer to achieve a given accuracy. The exception is when the gap is 0, and the smallest eigenvalue appears with multiplicity. In this case the algorithm actually requires fewer iterations.

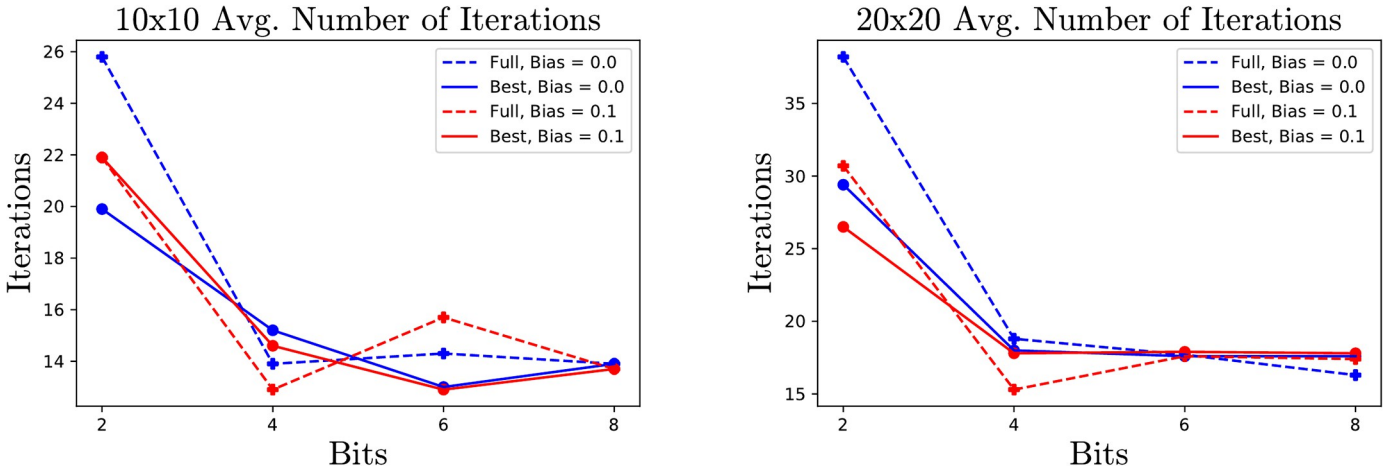

Fig 5. Average number of iterations required for MP matrices for different parameters.

https://doi.org/10.1371/journal.pone.0267954.g005

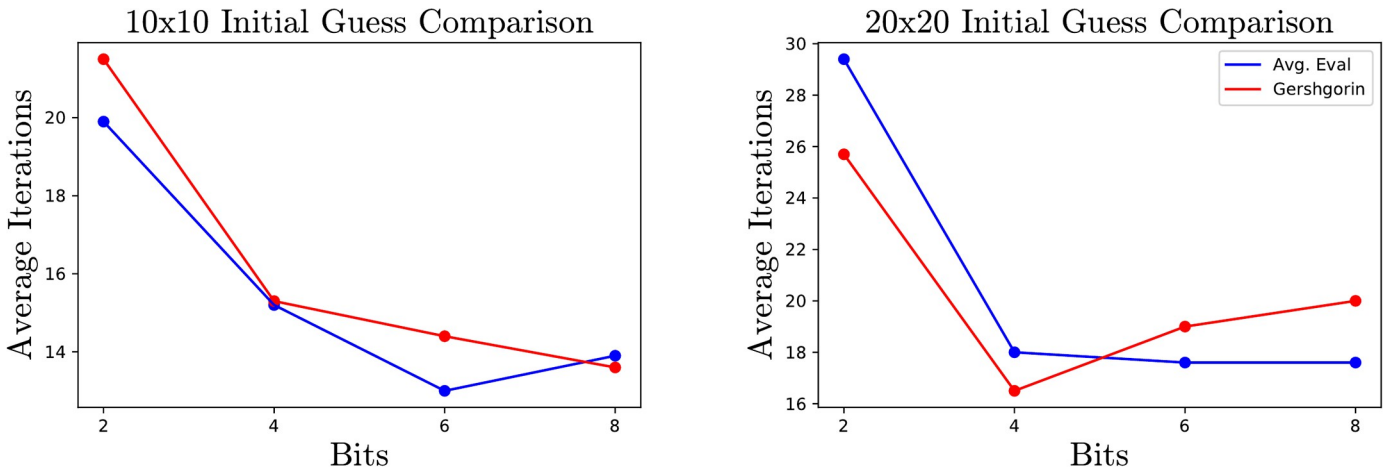

Fig 6. Total iterations required for different initializations of λ.

https://doi.org/10.1371/journal.pone.0267954.g006

Fig 8 has two example error plots demonstrating the slower convergence. Observe that the eigenvector error relative to both the precision and eigenvalue error increases as the gap size decreases.

In the case when there is degeneracy, that is the gap g is 0, one might want two eigenvectors that span the eigenspace. This can be accomplished by running the algorithm once to get an approximate eigenvector $\mathbf{v}_1$, replace the matrix A with $A + \alpha \mathbf{v}_1 \mathbf{v}_1^t$ where $\alpha > 0$, and run the algorithm again to get the eigenvector $\mathbf{v}_2$. By the spectral theorem for the symmetric matrix $A + \alpha \mathbf{v}_1 \mathbf{v}_1^t$, $\mathbf{v}_1^t \mathbf{v}_2 = 0$ implying that $\mathbf{v}_2$ is an eigenvector for A. Replacing A by $A + \alpha \mathbf{v}_1 \mathbf{v}_1^t$ is necessary for numeric purposes. The gap g is never numerically zero, so if the algorithm is run twice on the matrix A even with different randomization, it will often produce the same vector. One can also try to take advantage of the first computation by initializing the approximate eigenvalue to $\lambda_n + \epsilon$ in the second run of the algorithm. The data shown below in Fig 9 was collected for $\alpha = \frac{tr(A)}{n} - \lambda_n$ and $\epsilon = 1$, and one can see a slight boost in performance in the second run of the algorithm.

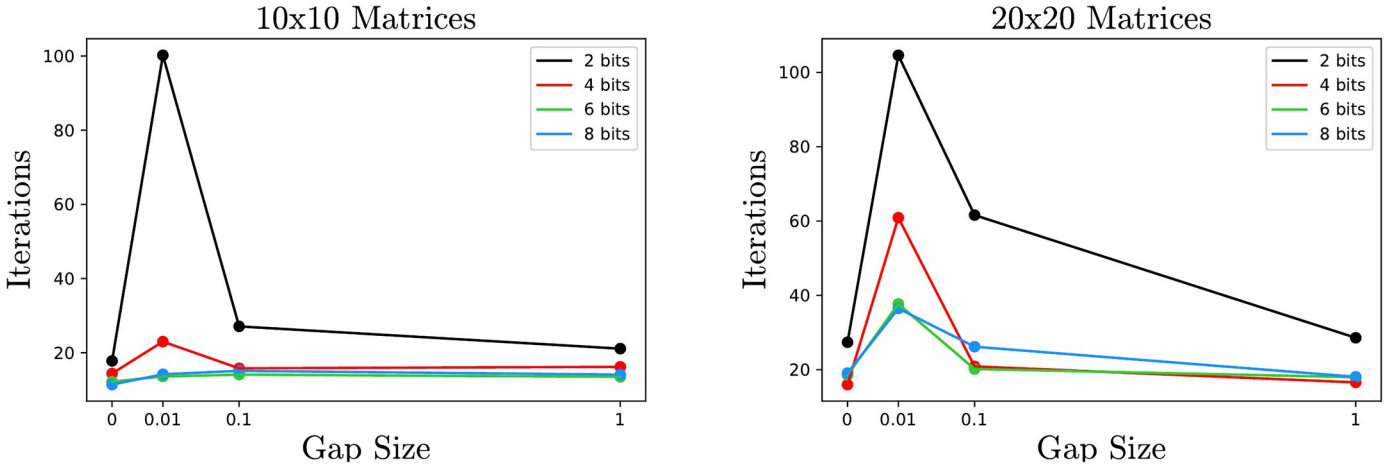

Fig 7. Average number of iterations on 30 samples as a function of the gap size $|\lambda_1 - \lambda_2|$. The smaller the gap, the more iterations required, especially when the number of bits is small. In the extreme case where the gap is 0 and the lowest eigenvalue appears with multiplicity, the algorithm is actually faster in the sense that fewer iterations are needed for the computed eigenvalue to approximate the true eigenvalue.

https://doi.org/10.1371/journal.pone.0267954.g007

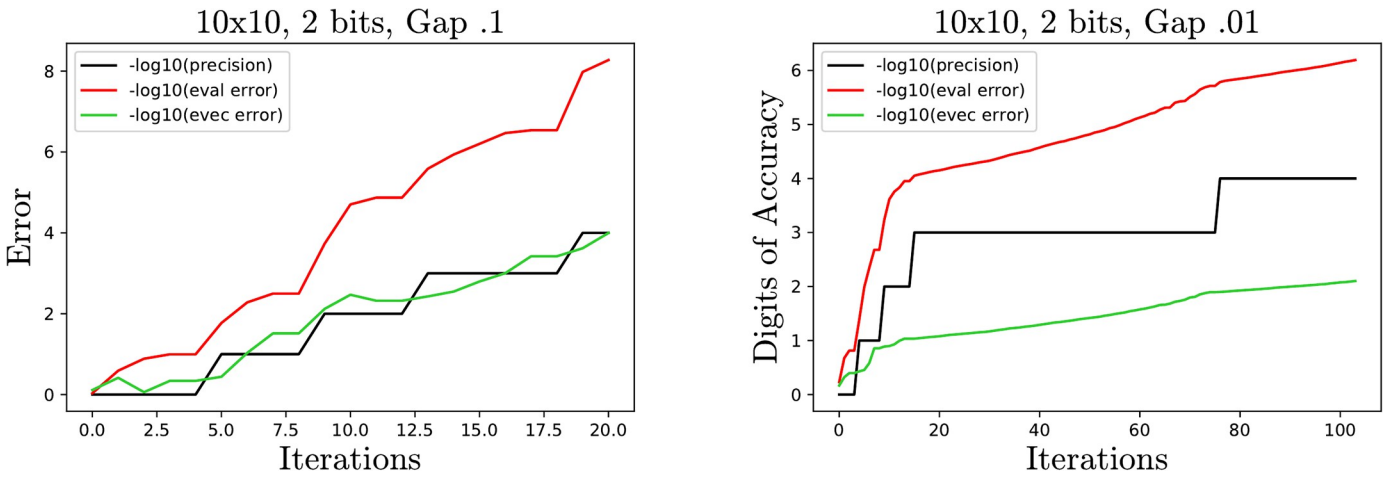

Fig 8. Sample plots for two 10 × 10 matrices with gap size .1 and .01. As the gap size decreases, the ratio of eigenvector error to precision and eigenvalue error increases.

https://doi.org/10.1371/journal.pone.0267954.g008

4 Conclusion

We have proposed and tested an algorithm to find eigenvectors of symmetric matrices by minimizing the corresponding Rayleigh quotient with an iterative steepest-descent method. Initial guesses and subsequent descent directions are found by looking for minima over discretized cubes of various sizes, encoded as QUBO problem which is in turn solved with a SA method. The algorithm is able to reach essentially arbitrary precision even for fairly large matrices. We have performed a thorough study of the effect of the different parameters, including, the eigenvalue spacing, initial guesses, number of bits, and the matrix size. We have explored the possibility of using a single bit precision by reformulating the QUBO problem as an Ising problem. Finally, we have introduced two novel approaches to accelerate the convergence such as biasing and using a larger set of solution from the SA step. These two approaches might be applicable to other QUBO based problems. We encourage the reader to test these algorithms on other annealing devices.

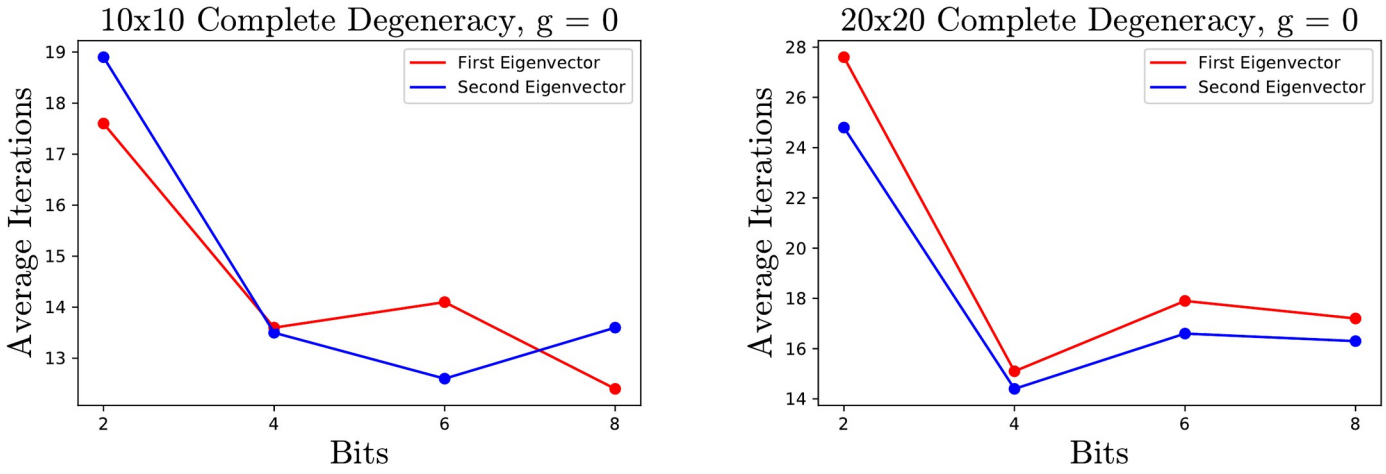

Fig 9. Average number of iterations running the algorithm twice on matrices with complete degeneracy. One can see slightly better performance on the second run, particularly on the 20 × 20 matrices.

https://doi.org/10.1371/journal.pone.0267954.g009

Acknowledgments

We acknowledge Los Alamos National Laboratory (LANL) for promoting and encuraging students Internships. Assigned: Los Alamos Unclassified Report 21-21969.

Author Contributions

Conceptualization: Susan M. Mniszewski, Christian F. A. Negre.

Formal analysis: Susan M. Mniszewski, Christian F. A. Negre.

Funding acquisition: Susan M. Mniszewski.

Investigation: Christian F. A. Negre.

Methodology: Benjamin Krakoff, Christian F. A. Negre.

Resources: Susan M. Mniszewski.

Software: Benjamin Krakoff, Christian F. A. Negre.

Supervision: Susan M. Mniszewski, Christian F. A. Negre.

Validation: Benjamin Krakoff, Susan M. Mniszewski, Christian F. A. Negre.

Visualization: Benjamin Krakoff.

Writing – original draft: Benjamin Krakoff.

Writing – review & editing: Susan M. Mniszewski, Christian F. A. Negre.

References

1. Page L, Brin S, Motwani R, Winograd T. The PageRank Citation Ranking: Bringing Order to the Web. Stanford InfoLab; 1999. 1999-66. Available from: http://ilpubs.stanford.edu:8090/422/.
2. Hall KM. An r-Dimensional Quadratic Placement Algorithm. Management Science. 1970; 17(3):219–229. https://doi.org/10.1287/mnsc.17.3.219
3. Pearson K. LIII. On lines and planes of closest fit to systems of points in space; 1901. Available from: https://doi.org/10.1080/14786440109462720.
4. Ushijima-Mwesigwa H, Negre CFA, Mniszewski SM. Graph Partitioning Using Quantum Annealing on the D-Wave System. In: Proceedings of the Second International Workshop on Post Moores Era Supercomputing. PMES'17. New York, NY, USA: Association for Computing Machinery; 2017. p. 22–29. Available from: https://doi.org/10.1145/3149526.3149531.
5. Chang CC, Gambhir A, Humble TS, Sota S. Quantum annealing for systems of polynomial equations. Scientific Reports. 2019; 9(1):10258. https://doi.org/10.1038/s41598-019-46729-0 PMID: 31311997
6. Kochenberger GA, Glover F, Alidaee B, Rego C. An Unconstrained Quadratic Binary Programming Approach to the Vertex Coloring Problem. Annals of Operations Research. 2005; 139(1):229–241. https://doi.org/10.1007/s10479-005-3449-7
7. Teplukhin A, Kendrick BK, Babikov D. Calculation of Molecular Vibrational Spectra on a Quantum Annealer. J Chem Theory Comput. 2019; 15(8):4555–4563. https://doi.org/10.1021/acs.jctc.9b00402 PMID: 31314517
8. Teplukhin A, Kendrick BK, Tretiak S, Dub PA. Electronic structure with direct diagonalization on a D-wave quantum annealer. Scientific Reports. 2020; 10(1). https://doi.org/10.1038/s41598-020-77315-4 PMID: 33247201
9. Rahman S, Lewis R, Mendicelli E, Powell S. SU(2) lattice gauge theory on a quantum annealer. 2021.
10. Stoer RBJ. Introduction to Numerical Analysis. Springer, New York, NY; 1980.
11. Barahona F. On the computational complexity of Ising spin glass models. Journal of Physics A Mathematical General. 1982; 15(10):3241–3253. https://doi.org/10.1088/0305-4470/15/10/028
12. Aramon M, Rosenberg G, Valiante E, Miyazawa T, Tamura H, Katzgraber HG. Physics-Inspired Optimization for Quadratic Unconstrained Problems Using a Digital Annealer. Frontiers in Physics. 2019; 7:48. https://doi.org/10.3389/fphy.2019.00048

13. Boixo S, Smelyanskiy VN, Shabani A, Isakov SV, Dykman M, Denchev VS, et al. Computational multi-qubit tunnelling in programmable quantum annealers. Nature Communications. 2016; 7(1):10327. https://doi.org/10.1038/ncomms10327 PMID: 26739797

14. Davis TA, Hu Y. The University of Florida Sparse Matrix Collection. ACM Trans Math Softw. 2011; 38 (1). https://doi.org/10.1145/2049662.2049663

15. Marčenko VA, Pastur LA. Distribution of Eigenvalues for Some Sets of Random Matrices. Mathematics of the USSR-Sbornik. 1967; 1(4):457–483. https://doi.org/10.1070/SM1967v001n04ABEH001994

16. Gershgorin S. Über die Abgrenzung der Eigenwerte einer Matrix. Bull Acad Sci URSS. 1931; 1931 (6):749–754.

17. Mezzadri F. How to generate random matrices from the classical compact groups. Notices of the American Mathematical Society. 2006; 54.