Building an iterative heuristic solver for a quantum annealer

Gili Rosenberg1 · Mohammad Vazifeh1 · Brad Woods1 · Eldad Haber2

Received: 29 August 2015 / Published online: 23 April 2016
© Springer Science+Business Media New York 2016

Abstract A quantum annealer heuristically minimizes quadratic unconstrained binary optimization (QUBO) problems, but is limited by the physical hardware in the size and density of the problems it can handle. We have developed a meta-heuristic solver that utilizes D-Wave Systems’ quantum annealer (or any other QUBO problem optimizer) to solve larger or denser problems, by iteratively solving subproblems, while keeping the rest of the variables fixed. We present our algorithm, several variants, and the results for the optimization of standard QUBO problem instances from OR-Library of sizes 500 and 2500 as well as the Palubeckis instances of sizes 3000–7000. For practical use of the solver, we show the dependence of the time to best solution on the desired gap to the best known solution. In addition, we study the dependence of the gap and the time to best solution on the size of the problems solved by the underlying optimizer. Our results were obtained by simulation, using a tabu 1-opt solver, due to the huge number of runs required and limited quantum annealer time availability.

✉ Gili Rosenberg
gili.rosenberg@1qbit.com
Mohammad Vazifeh
mmvazifeh@gmail.com
Brad Woods
brad.woods@1qbit.com
Eldad Haber
haber@math.ubc.ca

1 1QB Information Technologies (1QBit), 458–550 Burrard Street, Vancouver, BC V6C 2B5, Canada
2 Department of Mathematics and Earth and Ocean Science, University of British Columbia, Vancouver, BC V6T 1Z2, Canada
Keywords Quantum annealing · Quadratic unconstrained binary optimization · Combinatorial optimization · $k$-opt · Local search · Iterative heuristic solver

1 Introduction

1.1 The problem and previous work

A quadratic unconstrained binary optimization (QUBO) problem is defined by

$$\min x^T Q x$$

$$s.t. \quad x \in S,$$

where, without loss of generality, $Q \in \mathbb{R}^{N \times N}$ and $S$ represents the binary discrete set $\{0, 1\}^N$.

Many NP-hard combinatorial optimization problems arise naturally or can easily be reformulated as QUBO problems, such as the quadratic assignment problem, the maximum cut problem, the maximum clique problem, the set packing problem, and the graph colouring problem (see, for instance, Boros and Prékopa [1], Boros and Hammer [2], Bourjolly et al. [3], Du and Pardalos [4], Pardalos and Rodgers [5,6], Pardalos and Xue [7], and Kochenberger et al. [8]).

Numerous interesting applications expressed naturally in the form of a QUBO problem have appeared in the literature. Barahona et al. [9,10] formulate and solve the problem of finding exact ground states of spin glasses with magnetic fields. Alidaee et al. [11] study the problem of scheduling $n$ jobs non-preemptively on two parallel, identical processors to minimize weighted mean flow time as a QUBO problem. Bomze et al. [12] give a comprehensive discussion of the maximum clique (MC) problem. Included is the QUBO problem representation of the MC problem and a variety of applications from different domains. A QUBO problem has been used in the prediction of epileptic seizures [13]. Alidaee et al. [14] discuss the number partitioning problem as a QUBO problem.

To solve a QUBO problem, a number of exact methods have been developed [5,15–32]. However, due to the computational complexity of the problems, these approaches have only been able to solve small-sized instances. To obtain near-optimal solutions, several heuristic and metaheuristic approaches such as tabu search [33–41], simulated annealing [42,43], evolutionary algorithms [38,44–48], scatter search [49], GRASP [41,50], and iterated local search algorithms [51] have been proposed. Additionally, preprocessing techniques have been developed to simplify the solving of QUBO problems [52].

The majority of existing heuristic algorithms search relatively small neighbourhoods that can be searched efficiently. In this paper, we explore the possibility of utilizing a quantum annealer (see Sect. 1.2), which is able to search a portion of the $k$-flip neighbourhood of size $2^k$ in a single operation. We develop a hybrid algorithm that incorporates a quantum annealer into a classical algorithm and give experimental results that show that this approach could be competitive in the near future. Our results were obtained by simulation, using a tabu 1-opt solver, due to
the huge number of runs required and limited quantum annealer time availability.

The general idea of decomposing optimization problems in order to aid their solution is an old idea [53,54], and there has been some limited research focusing specifically on QUBO problems [55–57]. While working on this research we noted research on the possibility of solving larger QUBO problems by decomposing them into smaller problems and solving them on a D-Wave Systems quantum annealer [58]. Although the idea of “large-neighbourhood local search” is mentioned in brief in that research, which bears resemblance to the basic idea we cover in this paper, it is not developed in detail and no results are reported.

After the publication of our paper on the arXiv, we became aware of a paper which explores a similar idea [59]. The authors of that work explored the applicability of solution via local updates to two- and three-dimensional Ising spin glasses, as well as problems structured for the hardware graph of a D-Wave quantum annealer. Since those problems contain only very local couplings, even a simple approach showed success. Our contribution focuses on devising an algorithm that is effective for a more general class of problems, sparse or dense, which in general contain short- and long-range couplings.

1.2 The quantum annealer

D-Wave Systems has developed a scalable quantum annealer that solves QUBO problems, albeit with some limitations.

It has been argued that quantum annealing has an advantage over simulated annealing due to quantum tunnelling, which allows an optimizer to pass through barriers instead of going over them. This might provide a speedup for some problem classes [60–62]. Indeed, simulated quantum annealing [63–65] shows a speedup for some problem classes [63], but not for some others [66].

There is strong evidence that the D-Wave machine is quantum [67,68], but it is yet to be established conclusively whether it provides a quantum speedup over classical computers. There has been significant interest in benchmarking the D-Wave machines [69–73]. Most recently, Denchev et al. showed that for a specific class of problems, the D-Wave machine showed a $10^8$ speedup over a single classical core [74]. There is also an ongoing debate on how to define quantum speedup, and on which problems a quantum annealer could be expected to demonstrate it [75–77]. We expect results in the near future to shed light on these important points.

The connectivity of the D-Wave processor’s qubits is currently given by the Chimera graph [78]. Due to imperfections in the fabrication process, the processors contain faulty qubits and couplers which cause the actual hardware graph to be a subgraph of the ideal Chimera graph. Due to the sparsity of this hardware graph, it is often the case that the problem the user wishes to solve is of a higher density than the connectivity

1 The problem that the machine actually solves is known as an Ising model in the physics community, and can easily be transformed into the form $x^T Q x$, which is more common in the scientific computing community.
of the chip. In these cases, it is possible to identify multiple physical qubits with one logical qubit in order to increase the connectivity. The problem of finding a mapping from logical to physical qubits is known as graph minor-embedding (often referred to as “embedding” in this context). In particular, it is known that the largest complete graph that can be embedded on an ideal Chimera graph of size $n$ is of size $\sqrt{2n} + 1$ (for instance, for the current-generation 1152-qubit D-Wave 2X chip, the largest complete graph is of size 49) [79,80].

Due to noise in the system, the qubit biases and couplings for the problem that the quantum annealer actually solves can differ from those given by the user, which is known as “parameter misspecification”. The parameter misspecification error for the current-generation chip is 2–3.5% of the full parameter scale (in the qubit biases and couplings) [81]. The implication for the user is that the probability of finding the optimum solution is reduced as the noise is increased.

The remainder of this paper is organized as follows. In Sect. 2 we describe the algorithm. In Sect. 3 we explain the benchmarking methodology and present computational results. In Sects. 4 and 5 we discuss the results, the conclusions, and future work.

### 2 The general QUBO problem solver (GQS)

#### 2.1 The basic algorithm

We assume that we have an oracle that optimizes QUBO problems of size $k$, but our problem has $N \gg k$ variables. How can we use the oracle to solve this larger problem? Our idea, which we refer to as “convergence” henceforth, is to iteratively choose $k$ variables out of the $N$ variables, and optimize over those variables while keeping the other $N - k$ variables fixed, until the value of the objective function does not change for some number of steps. This basic idea can be improved upon significantly, by choosing intelligently which variables to update, for example, or using tabu search to disallow updating of the same variables too often, and so on (various improvements are described in Sect. 2.2). One can write a simple algorithm that uses this idea to find a local minimum for a problem of size $N$ (see Algorithm 1).

#### Algorithm 1 Basic Algorithm

1: procedure **BasicAlgorithm**  
2: \hspace{1em} $x \leftarrow$ a random $N$-bit configuration, $x \in \{0, 1\}^N$  
3: \hspace{1em} **while** not converged **do**  
4: \hspace{2em} Choose $k$ variables at random  
5: \hspace{2em} $x \leftarrow$ optimize over $k$ variables, given the fixed $N - k$ variables (using the underlying optimizer)  
6: \hspace{1em} **return** $x$

Each call to the underlying optimizer and the subsequent update will either decrease the objective function’s value or leave it constant, but is guaranteed not to increase

---

2 The chip is current as of April 2016.
it. This does not mean that the algorithm does not hill climb. It does, in fact, do so, but the hill climbing capability is contained within the optimizer of size $k$. This algorithm successfully finds a local minimum; however, we are searching for the global minimum, and hence we must improve the algorithm to improve our chances of success.

### 2.2 The improved algorithm

We here describe an improved version of Algorithm 1, which we refer to as the General QUBO Problem Solver (GQS). The pseudo-code is presented in Algorithm 2, and each of the components is described in greater detail below:

1. Better initial guesses (Sect. 2.3)
2. Different underlying optimizers (Sect. 2.4)
3. Escape strategies (Sect. 2.5)
4. Choice of variables (Sect. 2.6)
5. Tabu search for $k$-opt (Sect. 2.7)
6. 1-opt tabu search improvement (Sect. 2.8)
7. Stopping criteria and convergence criterion (Sect. 2.9)

Each of these components is added in a modular way, such that many variations on the algorithm can be accommodated within this same framework. The reasoning behind this is that we expect that for different classes of problems the best-performing variant of this algorithm will be different. In addition, the mode of use is important: the parameters would be different if what we were looking for is a quick solution as opposed to a high rate of success.

We note that the reason to keep track of the one-flip gains is that once they are initialized, they can be updated very efficiently [82,83], which allows one to both quickly update the objective function’s value after each $k$-variable update, and to use the one-flip gains to choose the variables to be updated (see Sect. 2.6.1).

### 2.3 Better initial guesses

We use the greedy heuristic given by Merz and Katayama [82] to find good (that is, with low value and diverse) starting configurations quickly. First, all variables are initialized to 0.5, then repeatedly flipped to either 0 or 1 depending on which update provides the largest one-flip gain in the objective function’s value, and then updating the one-flip gains and repeating, while updating each bit only once. The bits are initialized to 0.5 despite being binary (that is, having values of 0 or 1) since the idea is not to bias the configurations initially such that any of the bits will be more likely to be 0 or 1 (by the end of the process all of the bits are set to either 0 or 1).

When initializing the reference set (see Sect. 2.5.1), we experimented with using a deterministic version of this greedy heuristic for the first element in the reference...
Algorithm 2 Improved Algorithm

1: procedure IMPROVEDALGORITHM
2: \( V, x \leftarrow \text{InitialGuess}() \)
3: \( V_{\min}, x_{\min} \leftarrow V, x \)
4: \( C \leftarrow \{1, \ldots, N\} \)
5: \( T \leftarrow \{\} \)
6: \( \text{gains} \leftarrow \text{Calculate change in objective function value for flipping each bit} \)
7: while not Done() do
8: \( \text{ChosenVariables} \leftarrow \text{VariableChoice}(C, k) \)
9: \( x \leftarrow \text{Optimize over the } k \text{ ChosenVariables, given the fixed } N - k \text{ variables (using the underlying optimizer)} \)
10: Update one-flip gains and \( V \)
11: \( C, T \leftarrow \text{TabuUpdate()} \)
12: if \( V < V_{\min} \) then
13: \( V_{\min} \leftarrow V \)
14: \( x_{\min} \leftarrow x \)
15: if Converged() then
16: \( x \leftarrow \text{Escape()} \)
17: Update one-flip gains and \( V \)
18: return \( V_{\min}, x_{\min} \)

set (see Algorithm 4). In order to obtain a diverse reference set, for the remaining elements we used a randomized version, where a random bit is flipped randomly, and at each stage the best flip to 0 or 1 is chosen with a probability that depends on the best one-flip gains. Although this results in a diverse set of initial solutions, we found that starting with random configurations gave more diverse initial sets (for example, based on the average Hamming distance).

We note that the randomized version by Merz and Katayama [82] only works when the highest 1-flip gains for both the best 0-flip and the best 1-flip are positive (for example, if they are both negative, \( p \) can be negative). In addition, in that version the 1-flip gains are not updated after the initial random bit is flipped. We also note that our version is for minimization, not maximization. We present our corrected version of the pseudo-code; see Algorithm 3.

2.4 Different underlying optimizers

The proposed GQS algorithm lends itself to being solved using any appropriate optimizer, for example, an exhaustive search, a 1-opt tabu search, Gurobi Optimizer, CPLEX, or a quantum annealer. The algorithm has been successfully verified using the D-Wave Two quantum annealer. However, since annealing technology is at an early stage, limitations such as qubit connectivity, number of qubits, and noise and error levels restrict the size of problem that can be solved. For this reason, for our benchmarking we used a 1-opt tabu search, with a tabu tenure of 15–20 and convergence length of 10\( k \), where \( k \) is the size of the underlying optimizer. We have verified that the GQS algorithm works when using the quantum annealer as the underlying optimizer.
Algorithm 3 Randomized Greedy Initial Guess

1: procedure RANDOMIZEDGREEDY
2: \( C \leftarrow \{1, \ldots, N\} \)
3: for \( i \in \{1, \ldots, N\} \) do \( x_i = \frac{1}{2} \)
4: Calculate one-flip gains \( g_i \) for all \( i \in \{1, \ldots, N\} \)
5: Select \( k \in \{1, \ldots, N\} \) and \( l \in \{0, 1\} \) randomly and set \( x_k \leftarrow l \)
6: \( C \leftarrow C - \{k\} \)
7: repeat
8: Calculate one-flip gains \( g_i \) for all \( i \in C \)
9: Select \( k \in \{1, \ldots, N\} \) and \( l \in \{0, 1\} \) randomly and set \( x_k \leftarrow l \)
10: \( C \leftarrow C - \{k\} \)
11: until \( C = \{\} \)
12: return \( x \)

Algorithm 4 Deterministic Greedy Initial Guess

1: procedure DETERMINISTICGREEDY
2: \( C \leftarrow \{1, \ldots, N\} \)
3: for \( i \in \{1, \ldots, N\} \) do \( x_i = \frac{1}{2} \)
4: Calculate one-flip gains \( g_i \) for all \( i \in \{1, \ldots, N\} \)
5: repeat
6: Update one-flip gains \( g_i \) for all \( i \in C \)
7: Find \( k_0 \) with \( g_k^0 = \min_{i \in C} g_i^0 \) and \( k_1 \) with \( g_k^1 = \min_{i \in C} g_i^1 \)
8: if \( g_{k_0}^0 < 0 \) and \( g_{k_1}^1 \leq 0 \) then \( p \leftarrow \frac{g_{k_0}^0}{g_{k_0}^0 + g_{k_1}^1} \)
9: else if \( g_{k_0}^0 \geq 0 \) and \( g_{k_1}^1 > 0 \) then \( p \leftarrow \frac{g_{k_1}^1}{g_{k_0}^0 + g_{k_1}^1} \)
10: else if \( g_{k_0}^0 > 0 \) and \( g_{k_1}^1 < 0 \) then \( p \leftarrow 0 \)
11: else if \( g_{k_0}^0 < 0 \) and \( g_{k_1}^1 > 0 \) then \( p \leftarrow 1 \)
12: if random(0, 1) < \( p \) then
13: \( x_{k_0} \leftarrow 0; C \leftarrow C - \{k_0\} \)
14: \( x_{k_1} \leftarrow 1; C \leftarrow C - \{k_1\} \)
15: until \( C = \{\} \)
16: return \( x \)

2.5 Escape strategies

Once the local search phase converges, which consists of multiple calls to the underlying optimizer, the GQS attempts to “escape” by flipping some number of bits, and then performing another local search phase. If the solver converges to a different configuration, then the escape has succeeded in diversifying the search space of the algorithm. The most trivial example of an escape is to flip a fixed number of bits randomly. We describe some more sophisticated ways of escaping below.
2.5.1 Path relinking

Path relinking refers to maintaining a collection of the best solutions to which the solver converged, also referred to as a “reference” set or set of elite solutions, and then escaping to solutions found by fusing two elite solutions that have not been fused before. This method is used in Wang et al. [84], utilizing a tabu 1-opt local search.

Algorithm 5, “Path Relinking Escape”, is composed of two phases. In the first, we maintain the reference set given a new converged solution. If the converged solution already appears in the reference set, we discard it. Otherwise, if the reference set is not full we add the converged solution, and if the reference set is full we only add the solution if it is better than the worst solution in the reference set, which we then discard.

In the second phase, we perform the escape. If the reference set is full and unfused pairs remain with a sufficiently large Hamming distance, we choose an unfused pair randomly, and update the current solution to the fused “child”. If the reference set is not yet full, we escape to a random configuration. Once we run out of unfused pairs, we empty the reference set except for the best solution thus far found, and restart the procedure.

There are different ways to fuse the parents. We choose to ignore pairs of parents that have a Hamming distance $d$ less than a parent distance threshold (typically a small number such as 5). For parents with a Hamming distance greater than that, we identify the bits that are equal and the ones that are different in both parents. First, we set the equal bits in the child to be equal to those in the parents. Then we set the different bits in the child randomly, such that the resulting child’s Hamming distance to each of the parents is at least a user-defined fraction of the parent-parent Hamming distance (for example, 0.3).

### Algorithm 5 Path Relinking Escape

```plaintext
1: procedure PATHRELINKING_ESCAPE
2:   if $x \not\in$ ref_set then
3:     if ref_set not full then
4:       Add $x$ to ref_set
5:       Update unfused_pairs
6:     else
7:       if $V$ is better than worst in ref_set then
8:         Pop worst from ref_set
9:         Add $x$ to ref_set
10:        Update unfused_pairs
11:     end if
12:     if ref_set is full then
13:       if unfused_pairs is not empty then
14:         $x$ ← Fuse random pair from unfused_pairs
15:       else
16:         Empty ref_set except best
17:       end if
18:     end if
19:   else
20:     $x$ ← RandomizedGreedy()
21: end if
22: return $x$
```
2.5.2 f-smart method

We keep track of which variables were flipped in recent improvements. Then, when an escape is needed (in order to diversify the search space), we randomly flip some number of the most-flipped bits. The reasoning behind the flip frequency-based scheme is that certain variables are unstable (defined as being flipped often when updating), and those are precisely the variables we should flip to escape (as opposed to the more-stable variables). We also track the number of escapes with no update to \( V_{\text{min}} \), and increase the number of variables to flip as this number increases, making the flips increasingly drastic when less-drastic flips have failed to diversify the search space. This is a version of the adaptive memory idea introduced by Glover et al. [34,35].

2.6 Choice of variables

There are many ways to choose the \( k \) variables to solve for from amongst the \( \binom{N}{k} \) possibilities. The simplest one is to choose randomly. However, for all problems we tested there is an advantage to choosing more intelligently. For the benchmarked problems in this paper, a combination of gains-based and fusion-guided (see Sects. 2.6.1 and 2.6.2) choice of variables was the most advantageous.

2.6.1 Gains-based

In a 1-opt tabu search, the variable flipped is often the variable with the best one-flip gain. We generalize this such that we try to update the \( k \) variables with the best 1-opt gain. This method can get stuck easily if it is not used with tabu search for \( k \)-opt (see Sect. 2.7), since the \( k \) variables with the best \( k \) one-flip gains will be chosen repeatedly, even when no possible flip can be found that lowers the objective function’s value. Another possibility is to choose the \( k \) variables randomly, weighting the choice based on the one-flip gains.

2.6.2 Fusion-guided

This scheme is coupled with path relinking escapes (see Sect. 2.5.1). Once a pair of parents has been chosen, we can focus our search on the hypercube defined by the symmetric space between these parents. Let \( d \) be the Hamming distance between the parents. Then, if \( k \leq d \), we choose \( k \) variables randomly amongst the \( d \) variables; otherwise, we choose all of the \( d \) variables and \( k - d \) more variables from the set of variables that were equal in the parents. We do this for \( w \) iterations, before reverting back to one of the other variable-choice methods: gains, coupling, random, etc. In addition, we note that the tabu search for \( k \)-opt is not taken into account for these \( w \) iterations: the tabu list is kept empty and the candidate list is kept full.
2.6.3 Coupling-based

Intuitively, we might expect it to be advantageous to choose strongly correlated variables to be considered for an update together. There are many ways to do this, and the tradeoff between precision and computational time must be considered. We present a simple example; see Algorithm 6.

This algorithm iterates through two stages until finding $k$ variables out of the candidate variables $C$. In the first stage, a variable is chosen randomly, using probabilities derived from the sum of absolute values of coefficients in that variable’s column in the problem matrix $Q$ (we call this the “strength”). In the second stage, a variable that is connected (in the underlying adjacency matrix) to the variable in the first stage is chosen randomly, using probabilities derived from the absolute value of the coefficients in the first variable’s column (we call this the “conditional strengths”).

Algorithm 6 Coupling Choice Algorithm

```plaintext
1: procedure CouplingChoice
2: indices_to_update ← {} (an empty set)
3: $LC ← C$
4: $LQ ←$ project $Q$ on $C$
5: $n ← \text{dim}(LQ)$
6: for $i = 1$ to $n$ do $\text{strength}[i] ← \sum_j |LQ|_{ji}$
7: probabilities[$i] ← \text{strength}[i] / \text{sum(strength)}$, for each $i$
8: while length(indices_to_update) < $k$ do
9: $p ← \text{random}(0, 1)$ (first stage)
10: $i ← C[\text{index for interval in which } p \text{ falls in cumulative probabilities}]$
11: if $i ∈ LC$ then remove $i$ from $LC$ (exclude from second stage)
12: Add $i$ to indices_to_update
13: if length(indices_to_update) = $k$ then break
14: for $j ∈ LC$ do $\text{conditional_strengths}[j] ← |LQ|_{ji}$
15: if $\sum(\text{conditional_strengths}) = 0$ then continue
16: Divide conditional_probabilities by $\sum(\text{conditional_strengths})$
17: $p ← \text{random}(0, 1)$ (second stage)
18: $m ← LC[\text{index for interval in which } p \text{ falls in cumulative conditional_probabilities}]$
19: Remove $m$ from $LC$
20: Add $m$ to indices_to_update
21: return indices_to_update
```

2.6.4 Practical considerations for the quantum annealer

The specifics of the graph of the D-Wave chip are also a consideration. As stated in Sect. 1.2, for QUBO problems that are more dense than the hardware graph, it is necessary to identify multiple logical qubits with each physical qubit. The mapping between logical and physical qubits is a graph minor-embedding problem that is often referred to as an “embedding” in this context [79,80].

However, for sparse problems of size $N$, any subproblems of size $k$ are likely to be sparse as well. For this reason, in some scenarios it might be better not to use a complete embedding (forcing $k ≤ \sqrt{2n} + 1$, where $n$ is the number of qubits), since
it might be possible to embed considerably larger subproblems than that ($\sqrt{2n} + 1 < k \leq n$). The challenge is minimizing the time expense of finding an embedding for each subproblem.

Non-complete embeddings can be utilized in at least two ways:

1. Find a new embedding before each call. The disadvantage to this method is the time required to find an embedding, and the fact that one cannot know for sure if a graph can be embedded without trying to embed it. It may be possible to devise a heuristic that could judge which size and density combinations are likely to be embeddable, but multiple calls to an embedding finder might still be necessary before an embedding is found and can be used for a call to the quantum annealer.

2. Find a pool of subproblems and the corresponding embeddings in advance, and then choose one to use at each step. The advantage to this method is apparent for problems that share the same adjacency matrix, but the coefficients differ. One can then find a pool of subproblems and corresponding embeddings in advance, and at each stage choose one of the prechosen subproblems to optimize over. This method can be combined with the above methods; for example, after determining the $k$ variables with the best one-flip gains, we can choose the subproblem from the pool that contains the largest number of variables from the group of $k$ variables.

In the simulations included in this study, we assumed that a pre-computed complete embedding was used, hence no embedding time needs to be included in the timings.

2.7 Tabu search for $k$-opt

We define a version of tabu search for $k$-opt: we maintain a queue of length $TT$ (the tabu tenure), in which each element is a list of variables of length $k$. While the queue is not full we add a new group of $k$ variables at each update. Once the queue is full, we begin to also pop the oldest list of $k$ variables at each step. We also try a version of this in which only the variables that were actually flipped in the update are marked as tabu. In this case the elements in the queue are not of equal length, and indeed are sometimes empty (if no variable out of the $k$ was flipped in the update).

Unlike 1-opt tabu search, the objective of using the tabu tenure here is not to allow hill climbing, but rather to force the same variables to not be chosen for an update attempt too frequently. The reason for this is that when optimizing over the $k$ variables, at worst the same solution will be chosen for those $k$ variables (based on the way our algorithm is built). Hill climbing occurs from within the underlying optimizer: hills of up to $k$ flips can be climbed, in principle. Since most tabu search algorithms employ a tabu tenure of up to $\sim 20$, and typically $k > 20$, we consider this to be sufficient hill climbing capability.

2.8 1-opt tabu search improvement

We allow the option of 1-opt tabu search improvement phases immediately after an escape (before the $k$-opt phase), after convergence (after the $k$-opt phase), or with a probability $p$ of occurring instead of a $k$-opt update (disregarding the tabu queue of the
We typically set the tabu tenure low (or to 0) and the convergence length short, as the idea is to get a quick improvement. We motivate this improvement by noting that despite \( k\)-opt digging “deeper” than 1-opt, the latter has an advantage in that it is able to consider all of the variables at once for an update. When \( N \) is much larger than \( k \), each variable is visited less frequently by the GQS, and this advantage becomes increasingly important. We note that these optional tabu search 1-opt improvement phases are not indicated in the pseudo-code in the interest of clarity.

### 2.9 Stopping criteria and convergence criterion

For benchmarking purposes, we define stopping criteria as a maximum time limit or a maximum number of escapes to be performed. In addition, for some practical problems a known objective function’s value is sufficient, so we also allow stopping once a certain value is reached. For the convergence criterion, we define convergence as there being no update to the minimum value found since the last escape, within \( CL \) (convergence length) steps from the last escape. We also define an objective function’s tolerance \( tol \). Only prospective updates that lower the objective function’s value by more than \( tol \) are accepted.

### 3 Benchmarking

#### 3.1 Methodology

For benchmarking purposes, we simulated larger quantum annealers in order to make a claim regarding the scaling of the solution time and/or gap (that is, the difference in the objective function’s value from the best known solution). To do this, we replaced D-Wave’s machine’s being the underlying optimizer with a 1-opt tabu search with short-term memory and aspiration (see, for example, [35]), and then replaced the time the 1-opt tabu search takes with the expected time it would take the D-Wave solver to solve that problem (we chose 0.02 s). This allowed us to simulate a D-Wave machine that is many years away (at a significant cost in time). We have verified that the algorithm works with the D-Wave machine as the underlying optimizer, but due to delays as explained in Sect. 2.4, there are practical reasons why it is preferable to simulate the quantum annealer for our benchmarking.

The actual time it will take a new D-Wave machine to solve a problem of size \( k \) in the future is unknown, and it may be more or less than the number we used. This is due to advances in chip technology, and also depends on the way in which the machine will be used: it is possible to use a small number of reads, a short annealing time, and a short wait before readout to lower the time required. It is also possible that as the

---

4 We chose 0.02 s based on 1000 anneals each taking 20 \( \mu s \) (which is the current minimum anneal time). This is the actual time for computation, and does not include extra time for programming the chip, thermalization, etc. Since future run times are not known, this number is only meant as a rough estimate, to give an idea of the actual time the computation could take.

---

\( \square \) Springer
number of qubits grows, a longer annealing time may be required in order to maintain
the quality of results (this is expected once quantum annealers reach the threshold
where if they were to anneal any faster the results would become worse). For these
reasons, we also report the number of iterations (calls to the D-Wave machine) which
would remain accurate regardless of future run time. The total time for each call is
composed of a “quantum time”, which is the product of the number of iterations and
the presumed run time (see above), and a “classical time”, including all other solver
time before and after the underlying optimizer calls.

For each problem set and each parameter set, we solved each problem 32 times
with a time out of 90 s of wall-clock time for the problems of size 500, 2250 s for the
problems of size 2500, and $1.2N$ s for the problems of size 3000 and above. We report
the time to best solution, the average relative gap between the best solution found
compared to the best known solution, the number of underlying optimizer iterations
to the best solution, and the success rate (the percentage of repetitions that reached
the best known solution).

We benchmarked the GQS on three sets of problems. The first is the OR-Library
unconstrained binary quadratic programming problems [85]. We generated these prob-
lems (labelled “bqp” below) by randomly choosing integers between $-100$ and $+100$
from a uniform distribution such that the resulting problem has a density of 0.1 (that is,
only 10% of the elements are non-zero). They have been widely benchmarked in the
combinatorial optimization literature. We used the best known solutions from Tavares
[86].

The second set of problems, random fully dense QUBO problems (labelled “RFDQ”
below) was generated in the same way, but they have a density of 1. We obtained
the best known solutions by running a multi-start 1-opt tabu search solver with a
large number of iterations, as well as performing a second check by running the path
relinking algorithm of Wang et al. [84]. The third set of problems was generated using
the problem generator and settings written by Palubeckis5 [87] with densities ranging
from 0.5 to 1, and we used the best known solutions reported in Wang et al. [84].

All benchmarking was performed on Amazon Web Services using a c3.8xlarge
instance, which has 32 virtual CPUs running at 2800 MHz. Each call to the GQS was
run serially on a single CPU. The code was written in Python, utilizing optimized
external libraries such as Numpy and Scipy. An efficient implementation written in a
lower-level language such as C would give faster times. However, we expect our results
to still hold qualitatively, and much of the prospective time advantage of a lower-level
programming language is cancelled by our method of replacing the true run time of the
underlying optimizer with the assumed run time of the D-Wave annealer (the number
of iterations would remain the same).

### 3.2 Parameters

In the results below, we set the parameters that were not varied as listed in Table 1. The
settings were chosen empirically from problems of sizes up to 500 from OR-Library

---

5 The source code of the generator and the input files to create these problems can be found at http://www.
proin.ktu.lt/~gintaras/ubqop_its.html. This page was last retrieved on July 21, 2015.
Table 1  Benchmarking parameters (constant)

| Name                      | Value | Description                                                                 |
|---------------------------|-------|-----------------------------------------------------------------------------|
| Optimizer_TT              | 15–20 | Tabu tenure for the 1-opt tabu search underlying optimizer                  |
| Optimizer_CL              | 10k   | Maximum iterations with no improvement for the 1-opt tabu search optimizer  |
| Parent_distance_threshold | 5     | Minimum Hamming parent-parent distance                                       |
| Child_distance_threshold  | 0.33  | Minimum Hamming parent-child distance                                       |
| Num_elite_solutions       | 10    | Number of elite solutions to keep                                           |
| Variable_choice           | “Gains”| Variable choice type after w iterations (see below)                        |
| Presumed_time             | 0.02 s| Presumed run time for quantum annealer                                      |
| Tol                       | 10−8  | Minimum change in value for an update                                        |

Table 2  Benchmarking parameters (varying)

| Name   | Description                                         |
|--------|-----------------------------------------------------|
| k      | Variable group size for underlying optimizer        |
| CL     | Maximum calls to the underlying optimizer with no improvement |
| w      | Number of fusion-guided iterations                  |
| TT     | k-opt tabu tenure, typically \( N/k \)              |
| W      | Tabu whole group (True/False)                       |

as well as prior literature (such as [84]). The names and descriptions of the parameters that were varied are in Table 2.

In the case of a quantum annealer, \( k \) refers to the largest complete problem that can be solved with that quantum annealer. Depending on the physical connectivity of the chip, the number of qubits in the quantum annealer \( n \) could be much larger (see Sect. 1.2).

3.3 Results

3.3.1 OR-Library and RFDQ: 500 to 2500 variables

We present results for the OR-Library 500 and 2500 problems and RFDQ problems of size 500 below. We benchmarked the smaller instances comprehensively to choose the best settings, optimizing for highest success rate, which we then used for all remaining benchmarks. These runs were based on the path relinking escapes, with an initial reference set initialized from random, and updated using an underlying optimizer of size 50 with a convergence length of 3 and a \( k \)-opt tabu tenure of 6 (leading us to use \( TT = 0.6N/k \) hereafter), marking as tabu all variables chosen at each update (\( W = \) True). The variables chosen to be updated were based on the fusion-guided approach with \( w = 1 \), and based on the best one-flip gains thereafter (and during the creation of the reference set). The underlying optimizer was 1-opt tabu search with a tabu tenure
Table 3 Results for size 500 for $k = 50$, $CL = 3$, $TT = 64$, $w = 1$, fusion-guided path relinking with $W = True$

| Problem set | $\langle T \rangle$ | STD($T$) | $\langle G \rangle$ | STD($G$) | Succ. | $\langle I \rangle$ | STD($I$) |
|-------------|---------------------|-----------|---------------------|----------|-------|---------------------|----------|
| bq500       | 4.83                | 2.60      | 0.02                | 0.06     | 60.62 | 158.3               | 91.2     |
| RFDQ500     | 2.98                | 2.50      | 0.01                | 0.04     | 78.75 | 95.9                | 87.8     |

The average time to best solution in seconds (assuming an underlying optimizer call time of 0.02 s) is denoted $\langle T \rangle$, the average gap to the best known solution as a percentage is denoted $\langle G \rangle$, the fraction of GQS calls that resulted in finding the best known solution is denoted “Succ.”, and the average number of calls to the underlying optimizer is denoted $\langle I \rangle$.

Table 4 Detailed results for OR-Library 500 for $k = 50$, $CL = 3$, $TT = 6$, $w = 1$, fusion-guided path relinking with $W = True$

| Name       | $\langle T \rangle$ | STD($T$) | $\langle G \rangle$ | STD($G$) | Succ. | $\langle I \rangle$ | STD($I$) |
|------------|---------------------|-----------|---------------------|----------|-------|---------------------|----------|
| bq500-1    | 5.85                | 2.51      | 0.10                | 0.13     | 50.00 | 197.3               | 87.9     |
| bq500-2    | 3.74                | 2.54      | 0.00                | 0.01     | 90.62 | 118.3               | 88.3     |
| bq500-3    | 1.55                | 1.00      | 0.00                | 0.00     | 100.00| 45.8                | 33.7     |
| bq500-4    | 5.93                | 1.89      | 0.01                | 0.01     | 34.38 | 188.2               | 67.1     |
| bq500-5    | 3.52                | 1.82      | 0.00                | 0.02     | 96.88 | 113.7               | 64.1     |
| bq500-6    | 6.25                | 1.80      | 0.04                | 0.04     | 46.88 | 206.3               | 64.2     |
| bq500-7    | 6.35                | 2.10      | 0.02                | 0.02     | 37.50 | 209.4               | 75.6     |
| bq500-8    | 6.34                | 2.10      | 0.04                | 0.02     | 12.50 | 211.5               | 76.7     |
| bq500-9    | 5.96                | 1.99      | 0.02                | 0.04     | 40.62 | 204.9               | 71.1     |
| bq500-10   | 2.85                | 1.75      | 0.01                | 0.06     | 96.88 | 87.2                | 58.7     |

See Table 3 for a definition of $\langle T \rangle$, $\langle G \rangle$, “Succ.”, and $\langle I \rangle$.

of 15 and convergence length 500. We verified that the algorithm is not very sensitive to the exact parameter values chosen: the differences are small, and are close to the statistical variation of the results.

Table 3 includes the results for each problem set, averaged over the 10 problems in each set, as well as 32 repetitions for each problem. Detailed results for each problem (averaged over the 32 repetitions) in the OR-Library and RFDQ problem sets are presented in Tables 4 and 5, respectively.

In addition, Table 6 includes the results for OR-Library 2500, averaged over the 10 problems in each set, as well as 32 repetitions for each problem. Detailed results for each problem (averaged over the 32 repetitions) in the OR-Library problem sets are presented in Table 7.

3.3.2 Palubeckis: 3000 to 7000 variables

Results for the Palubeckis instances of sizes 3000–7000 are presented in Table 8.

3.3.3 Summary of all results: 500 to 7000 variables

We summarize all of the results for average time and average gap in Fig. 1.
Table 5  Detailed results for RFDQ 500 for \( k = 50, CL = 3, TT = 8, w = 1 \), fusion-guided path relinking with \( W = True \)

| Name       | \( \langle T \rangle \) | STD(\( T \)) | \( \langle G \rangle \) | STD(\( G \)) | Succ. | \( \langle I \rangle \) | STD(\( I \)) |
|------------|----------------|-------------|----------------|-------------|-------|----------------|-------------|
| RFDQ500-1  | 3.91           | 2.59        | 0.08           | 0.03        | 12.50 | 129.7         | 91.1        |
| RFDQ500-2  | 4.71           | 2.94        | 0.01           | 0.01        | 62.50 | 156.8         | 104.5       |
| RFDQ500-3  | 1.25           | 0.77        | 0.00           | 0.00        | 100.00| 36.2          | 24.1        |
| RFDQ500-4  | 3.74           | 1.95        | 0.03           | 0.06        | 75.00 | 121.7         | 67.2        |
| RFDQ500-5  | 1.52           | 1.01        | 0.00           | 0.00        | 100.00| 44.7          | 33.1        |
| RFDQ500-6  | 5.03           | 2.69        | 0.02           | 0.05        | 84.38 | 173.2         | 95.3        |
| RFDQ500-7  | 3.07           | 2.62        | 0.01           | 0.01        | 68.75 | 95.5          | 89.1        |
| RFDQ500-8  | 1.26           | 0.76        | 0.00           | 0.00        | 100.00| 34.6          | 23.2        |
| RFDQ500-9  | 3.94           | 2.43        | 0.01           | 0.01        | 84.38 | 128.7         | 86.2        |
| RFDQ500-10 | 1.32           | 0.86        | 0.00           | 0.00        | 100.00| 37.8          | 28.2        |

See Table 3 for a definition of \( \langle T \rangle \), \( \langle G \rangle \), “Succ.”, and \( \langle I \rangle \)

Table 6  Results for size 2500 for \( k = 50, CL = 3, TT = 30, w = 1 \), fusion-guided path relinking with \( W = True \)

| Identifier | \( \langle T \rangle \) | STD(\( T \)) | \( \langle G \rangle \) | STD(\( G \)) | Succ. | \( \langle I \rangle \) | STD(\( I \)) |
|------------|----------------|-------------|----------------|-------------|-------|----------------|-------------|
| bq2500     | 173.07         | 90.76       | 0.02           | 0.02        | 34.06 | 3798.0        | 2008.6      |

See Table 3 for a definition of \( \langle T \rangle \), \( \langle G \rangle \), “Succ.”, and \( \langle I \rangle \)

Table 7  Detailed results for RFDQ 500 for \( k = 50, CL = 3, TT = 30, w = 1 \), fusion-guided path relinking with \( W = True \)

| Name       | \( \langle T \rangle \) | STD(\( T \)) | \( \langle G \rangle \) | STD(\( G \)) | Succ. | \( \langle I \rangle \) | STD(\( I \)) |
|------------|----------------|-------------|----------------|-------------|-------|----------------|-------------|
| bq2500-1   | 180.63         | 82.01       | 0.01           | 0.01        | 34.38 | 3966.6        | 1804.7      |
| bq2500-2   | 197.50         | 95.32       | 0.05           | 0.02        | 0.00  | 4381.8        | 2091.1      |
| bq2500-3   | 190.15         | 74.70       | 0.03           | 0.02        | 9.38  | 4240.1        | 1679.0      |
| bq2500-4   | 77.32          | 49.22       | 0.00           | 0.00        | 100.00| 1634.7        | 1070.9      |
| bq2500-5   | 140.69         | 87.81       | 0.00           | 0.00        | 62.50 | 3033.3        | 1915.4      |
| bq2500-6   | 198.85         | 80.40       | 0.01           | 0.01        | 25.00 | 4334.7        | 1761.6      |
| bq2500-7   | 245.32         | 67.62       | 0.03           | 0.02        | 18.75 | 5387.8        | 1472.5      |
| bq2500-8   | 170.00         | 85.64       | 0.01           | 0.01        | 46.88 | 3716.1        | 1866.9      |
| bq2500-9   | 148.07         | 83.10       | 0.01           | 0.01        | 43.75 | 3241.9        | 1854.1      |
| bq2500-10  | 182.20         | 88.32       | 0.05           | 0.02        | 0.00  | 4043.4        | 1965.8      |

See Table 3 for a definition of \( \langle T \rangle \), \( \langle G \rangle \), “Succ.”, and \( \langle I \rangle \)

3.3.4 Benchmarking for practical requirements

For practical use, we expect that for many use cases, once a solution has been found that is close enough to the global optimum, one can stop the search based on stopping criteria, even if the confidence in having successfully found the global optimum is not
Table 8  Detailed results for Palubeckis instances of size 3000–7000 for $k = 50$, $CL = 3$, $w = 1$, fusion-guided path relinking with $W = \text{True}$

| Name    | $(T)$  | STD$(T)$ | $(G)$ | STD$(G)$ | Succ. | $(I)$ | STD$(I)$ |
|---------|--------|----------|-------|----------|-------|-------|----------|
| p3000-1 | 292.89 | 135.37   | 0.03  | 0.03     | 40.62 | 5967.0| 2770.8   |
| p3000-2 | 280.17 | 124.55   | 0.01  | 0.01     | 18.75 | 5614.8| 2486.6   |
| p3000-3 | 313.99 | 153.08   | 0.03  | 0.02     | 15.62 | 6367.3| 3080.9   |
| p3000-4 | 368.17 | 148.97   | 0.04  | 0.01     | 3.12  | 7423.2| 2978.1   |
| p3000-5 | 323.24 | 142.11   | 0.03  | 0.01     | 0.00  | 6493.5| 2843.6   |
| p3000   | 315.69 | 144.39   | 0.03  | 0.02     | 15.62 | 6373.2| 2903.9   |
| p4000-1 | 599.18 | 143.46   | 0.02  | 0.03     | 43.75 | 10431.8| 2619.2   |
| p4000-2 | 466.75 | 187.62   | 0.05  | 0.02     | 6.25  | 8018.7| 3202.9   |
| p4000-3 | 544.72 | 151.48   | 0.04  | 0.02     | 0.00  | 9430.6| 2619.2   |
| p4000-4 | 403.19 | 210.82   | 0.02  | 0.02     | 40.62 | 6996.3| 3650.3   |
| p4000-5 | 564.33 | 184.65   | 0.06  | 0.02     | 0.00  | 9680.3| 3147.6   |
| p4000   | 515.63 | 191.05   | 0.04  | 0.03     | 18.12 | 8911.5| 3290.8   |
| p5000-1 | 773.82 | 276.58   | 0.06  | 0.03     | 0.00  | 11708.2| 4163.0   |
| p5000-2 | 614.80 | 263.88   | 0.04  | 0.02     | 0.00  | 9254.3| 3917.7   |
| p5000-3 | 623.26 | 251.04   | 0.07  | 0.03     | 3.12  | 9459.2| 3739.8   |
| p5000-4 | 783.66 | 228.80   | 0.07  | 0.03     | 0.00  | 11714.3| 3402.6   |
| p5000-5 | 774.58 | 211.93   | 0.05  | 0.02     | 0.00  | 11426.0| 3088.7   |
| p5000   | 714.02 | 259.46   | 0.06  | 0.03     | 0.00  | 10712.4| 3846.7   |
| p6000-1 | 1018.34| 302.98   | 0.08  | 0.04     | 0.00  | 13584.2| 3974.2   |
| p6000-2 | 944.05 | 246.74   | 0.06  | 0.03     | 0.00  | 12556.8| 3255.0   |
| p6000-3 | 955.48 | 293.04   | 0.06  | 0.03     | 0.00  | 12813.8| 3909.6   |
| p6000   | 972.62 | 283.87   | 0.07  | 0.03     | 0.00  | 12985.0| 3752.6   |
| p7000-1 | 1215.57| 362.55   | 0.06  | 0.02     | 0.00  | 14539.8| 4285.1   |
| p7000-2 | 1209.88| 371.84   | 0.07  | 0.02     | 0.00  | 14574.3| 4373.1   |
| p7000-3 | 1179.84| 337.17   | 0.09  | 0.02     | 0.00  | 14225.0| 3940.6   |
| p7000   | 1201.76| 357.83   | 0.07  | 0.02     | 0.00  | 14446.4| 4206.7   |

See Table 3 for a definition of $(T)$, $(G)$, “Succ.”, and $(I)$

high. The user can define how close is sufficient. With this in mind, we can run the solver such that it stops once we find a solution within a given gap to the best known solution. In practice, the average time (and distribution of times) necessary to do this for a given set of problems could then be used as a guideline for the time it would require to attain that level of quality of solutions to new problems (assuming they are of similar difficulty).

As an example, we consider that for many applications, once a local optimum has been found with a gap of, for example, 0.1%, there is little benefit in searching any further. One reason for this is that the parameters of the problem often have considerable uncertainties associated with them, leading to an uncertainty in the objective function’s value that is larger than the gap itself. In other applications, such as machine learning, it is undesirable to “over-fit”, by finding the very best fit of the model to the
We investigated how the time required varied as we change the desired gap. Figure 2 provides results for the OR-Library and RFDQ problems of size 500. For these problems, we found that the time required to find a solution that is, for instance, 1% from the best known solution is low. In addition, as we require finding a solution closer to the best known solution, the time rises rapidly, as does the variation in the time required to find said solution, for both types of problems.

### 3.3.5 Dependence on size of underlying optimizer

We investigated the dependence of the average time to best solution on the underlying optimizer size; see Fig. 3a. We found that the time to best solution falls exponentially,
Building an iterative heuristic solver for a quantum annealer

Fig. 3 a Average time to best solution as a function of $k$, the simulated D-Wave chip size. Data was collected for GQS with parameters $CL=3$, $w=1$, fusion-guided path relinking with $W=True$ for OR-Library problems of size 2500, averaged over the ten problems and 32 repetitions, and $TT$ was chosen such that the tabu list is 60% of the problem size. b Log of the average time to best solution as a function of the underlying optimizer size (same data as (a))

Fig. 4 Average gap as a function of the simulated D-Wave chip size, with the same parameters as in Fig. 3

as can be seen by observing that the time to best solution falls linearly as a function of the logarithm of the underlying optimizer size; see Fig. 3b. The expectation is that as $k \rightarrow N$, the number of iterations required should go to 1, giving a time equal to the presumed time for one call to the underlying optimizer (in this case 0.02 s). The exponential decrease can be understood by observing that as the underlying optimizer increases in size, it is able to scan an exponentially increasing variable space in constant time (based on our assumption of constant time).

We also investigated the dependence of the average gap obtained on the underlying optimizer size; see Fig. 4. We see that the gap decreases slowly. Intuitively, as the underlying optimizer size is increased, the GQS considers flipping larger and larger subgroups of bits, which contain within them the bit flips for any smaller underlying optimizers. Hence, we expect the gap to stay constant at worst, and at best to improve.
Table 9  1-opt tabu results

| Set     | ⟨T⟩ | STD(T) | ⟨G⟩ | STD(G) | Succ. | ⟨I⟩ | STD(I) |
|---------|-----|--------|-----|--------|-------|-----|--------|
| bqp500  | 0.30| 0.19   | 0.02| 0.02   | 52.00 | 1347| 974    |
| RFDQ500 | 0.38| 0.30   | 0.00| 0.01   | 95.00 | 1837| 1596   |

See Table 3 for a definition of ⟨T⟩, ⟨G⟩, and “Succ”. The average number of one-bit flips is denoted ⟨I⟩

4 Discussion

We have also tried solving the two sets of problems bqp500 and RFDQ500 with a 1-opt tabu search with a tabu tenure of 20 and a convergence length of 2500 with 100 repetitions per problem (see Table 9). The results show that the time to best solution is only slightly higher for the dense problems (recall that the OR-Library problems have a density of 0.1), but the success rate is much higher and the gap is lower. Based on this, finding a good local optimum appears to be similarly hard in both cases (for tabu 1-opt search), but finding the global optimum is harder in the sparse problems. For most practical applications, we expect the difference between the gaps (0.02 and < 0.005 %) to be negligible, so for practical purposes it appears that these two sets of problems are of similar difficulty.

We note that one might have expected that the dense problems would be harder, but they do not appear to be. To explain this, we hypothesize that sparse random problems contain many more local minima (relative to the one-flip neighbourhood), and hence the optimizer can more easily get stuck in a local minimum and miss the global optimum. A possible explanation for this is that given a solution and its objective function’s value, in a sparse problem the probability of flipping some number of bits and ending up at a similar objective function’s value is much higher than in a dense problem. It follows that the probability of the existence of multiple local minima with a similar value of the objective function is much greater.

Comparing the results of GQS in Table 3 (where \( k = 50 \)) with the results for the 1-opt tabu search in Table 9, we note first that the GQS results are marginally better for the OR-Library problems (that is, they have a higher success rate) and marginally worse for the RFDQ problems. Secondly, we note that the time to best solution is considerably longer for the GQS for both sets of problems.

As we increase the underlying optimizer’s size, we expect the average time to best solution to drop and the quality of the solution (that is, the average gap) to either decrease or remain constant. Based on this expectation, we predict that when the underlying optimizer’s (the D-Wave chip) size is large enough, it will eventually beat 1-opt tabu search and all other classical solvers (see Sect. 3.3.5). Exactly when this occurs will depend on the quality and run time of the D-Wave processor.

An attempt to compare these results directly with the best known classical algorithms is, unfortunately, difficult. For example, Wang et al.’s path relinking algorithm solves the OR-Library 2500 problems in \( \sim 40 \) s with high confidence in finding the optimum on a single-core Intel Pentium processor running at 2.83 GHz [84]. A com-
comparison with Fig. 3a to find when our algorithm would be expected to have a similar
time to best solution, gives an estimate of $k \approx 150$ (assuming that the solution quality
is similar). We note that the usefulness of this type of comparison is quite limited as
it compares the D-Wave chip to a single core and assumes that the annealing time
remains constant.

5 Conclusions and future work

We have shown that it is possible to use the D-Wave machine to heuristically solve sig-
nificantly larger problems than the chip intrinsically allows. If in the future a D-Wave
machine is able to optimize QUBO problems faster than the best classical algorithms,
this contribution could make the machine more suitable for solving real-world prob-
lems earlier. Based on our approach, we found that sparse and dense random problems
of size 500 can be optimized using a $k = 50$ simulated quantum annealer to within
0.02% of the optimum in 100–160 iterations, assuming the quantum annealer’s solution
quality is similar to 1-opt tabu search. If we assume that the run time is 0.02 s,
solving these problems requires 3–5 s. Keeping the solver size constant, for larger
problem sizes, the number of iterations increases rapidly and the quality of the pro-
duced solutions degrades, although it remains within 0.07% for all problems, even for
the dense problems of size 7000 (and would likely reduce further if we increased the
time out). We would expect these results to improve when using a better underlying
optimizer (that is, one with a higher success rate).

Although the time to best solution for a D-Wave chip with a complete graph of
size $k = 50$ is not competitive, we have shown that as $k$ grows, the time required to
solve a given problem drops exponentially, assuming constant call time to the quantum
annealer. Based on this assumption, and assuming quantum annealers scale faster than
classical algorithms on classical hardware, we would expect that the D-Wave chip will
eventually beat 1-opt tabu search and all other classical solvers. Exactly when this will
occur will depend on the quality and run time of the processor.

Our estimate of the time that will be required in the future for a D-Wave machine
to solve different problems has an unknowable precision since it depends on future
advances in engineering which could change the run time of the machine. Simi-
larly, future advances in error correction [88–92] and expected decreases in intrinsic
error will affect the quality of the quantum annealer’s results. To the extent that
this quality is better than the underlying optimizer we used to simulate the D-Wave
machine, the actual D-Wave machine’s solving time could decrease faster than esti-
mated.

One way in which our algorithm could be improved is by implementing prepro-
cessing. There are many ways in which QUBO problems can be preprocessed to decrease
the size of the problem (several are reviewed in [86]). In addition, using a better under-
lying optimizer such as the path relinking algorithm would undoubtedly improve the
results [84]. Finally, we look forward to improvements in the D-Wave machine in
the coming years, at which point we hope to see competitive results that use the
machine itself as the underlying optimizer (and possibly using the embedding ideas
from Sect. 2.6.4).
Acknowledgments The authors would like to thank Marko Bucyk for editing a draft of this paper and Robyn Foerster, Phil Goddard, and Pooya Ronagh for their useful comments. This work was supported by 1QB Information Technologies (1QBit) and Mitacs.

Compliance with ethical standards

Conflicts of interest EH declares no conflict of interest. GR, BW, and MV were academic interns at 1QBit when the work was done, and GR and BW are still at 1QBit. 1QBit is focused on solving real-world problems using quantum computers. D-Wave Systems is a minority investor in 1QBit.

References

1. Boros, E., Prékopa, A.: Probabilistic bounds and algorithms for the maximum satisfiability problem. Ann. Oper. Res. 21(1–4), 109–126 (1989)
2. Boros, E., Hammer, P.L.: Pseudo-Boolean optimization. Discrete Appl. Math. 123(1–3) pp. 155–225 (2002). Workshop on Discrete Optimization, DO’99, Piscataway
3. Bourjolly, J.-M.: A quadratic 0-1 optimization algorithm for the maximum clique and stable set problems. Technical Report University of Michigan, Ann Arbor (1994)
4. Du, D.-Z., Pardalos, P.M.: Handbook of Combinatorial Optimization: Supplement, vol. 1. Springer, New York (1999)
5. Pardalos, P.M., Rodgers, G.P.: Computational aspects of a branch and bound algorithm for quadratic zero-one programming. Computing 45(2), 131–144 (1990)
6. Pardalos, P.M., Rodgers, G.P.: A branch and bound algorithm for the maximum clique problem. Comput. Oper. Res. 19(5), 363–375 (1992)
7. Pardalos, P.M., Xue, J.: The maximum clique problem. J. Global Optim. 4(3), 301–328 (1994)
8. Kochenberger, G.A., Glover, F., Alidaee, B., Rego, C.: A unified modeling and solution framework for combinatorial optimization problems. OR Spect. 26(2), 237–250 (2004)
9. Barahona, F., Grötschel, M., Jünger, M., Reinelt, G.: An application of combinatorial optimization to statistical physics and circuit layout design. Oper. Res. 36(3), 493–513 (1988)
10. De Simone, C., Diehl, M., Jünger, M., Mutzel, P., Reinelt, G., Rinaldi, G.: Exact ground states of ising spin glasses: New experimental results with a branch-and-cut algorithm. J. Stat. Phys. 80(1–2), 487–496 (1995)
11. Alidaee, B., Kochenberger, G.A., Ahmadian, A.: 0–1 quadratic programming approach for optimum solutions of two scheduling problems. Int. J. Syst. Sci. 25(2), 401–408 (1994)
12. Bomze, I.M., Budinich, M., Pardalos, P.M., Pelillo, M.: The maximum clique problem. Handbook of Combinatorial Optimization, pp. 1–74. Springer, New York (1999)
13. Iasemidis, L.D., Pardalos, P., Sackellaes, J.C., Shiau, D.-S.: Quadratic binary programming and dynamical system approach to determine the predictability of epileptic seizures. J. Comb. Optim. 5(1), 9–26 (2001)
14. Alidaee, B., Glover, F., Kochenberger, G.A., Rego, C.: A new modeling and solution approach for the number partitioning problem. J. Appl. Math. Decis. Sci. 2005(2), 113–121 (2005)
15. Gulati, V., Gupta, S., Mittal, A.: Unconstrained quadratic bivalent programming problem. Eur. J. Oper. Res. 15(1), 121–125 (1984)
16. Carter, M.W.: The indefinite zero-one quadratic problem. Discr. Appl. Math. 7(1), 23–44 (1984)
17. Williams, H.P.: Model Building in Linear and Integer Programming. Springer, Berlin (1985)
18. Barahona, F., Jünger, M., Reinelt, G.: Experiments in quadratic 0–1 programming. Math. Prog. 44(1–3), 127–137 (1989)
19. Pardalos, P.M., Rodgers, G.P.: Parallel branch and bound algorithms for quadratic zero–one programs on the hypercube architecture. Ann. Oper. Res. 22(1), 271–292 (1990)
20. Billionnet, A., Sutter, A.: Minimization of a quadratic pseudo-boolean function. Eur. J. Oper. Res. 78(1), 106–115 (1994)
21. Palubeckis, G.: A heuristic-based branch and bound algorithm for unconstrained quadratic zero-one programming. Computing 54(4), 283–301 (1995)
22. Helmberg, C., Rendl, F.: Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. Math. Prog. 82(3), 291–315 (1998)
23. Hansen, P., Jaumard, B., Meyer, C., Groupe, Q.: d’études et de recherche en analyse des décisions (Montréal, Exact sequential algorithms for additive clustering. Montréal: Groupe d’études et de recherche en analyse des décisions (2000)

24. Huang, H.-X., Pardalos, P., Prokopyev, O.: Lower bound improvement and forcing rule for quadratic binary programming. Comput. Optim. Appl. 33(2–3), 187–208 (2006)

25. Pardalos, P.M., Prokopyev, O.A., Busygin, S.: Continuous approaches for solving discrete optimization problems. In: Handbook on Modelling for Discrete Optimization, pp. 39–60, Springer (2006)

26. Pan, S., Tan, T., Jiang, Y.: A global continuation algorithm for solving binary quadratic programming problems. Comput. Optim. Appl. 41(3), 349–362 (2008)

27. Gueye, S., Michelon, P.: A linearization framework for unconstrained quadratic (0–1) problems. Discret. Appl. Math. 157(6), 1255–1266 (2009). Reformulation Techniques and Mathematical Programming

28. Pham Dinh, T., Nguyen Canh, N., Le Thi, H.: An efficient combined dca and bnb using dc/sdp relaxation for globally solving binary quadratic programs. J. Global Optim. 48(4), 595–632 (2010)

29. Mauri, G.R., Lorena, L.A.N.: Lagrangean decompositions for the unconstrained binary quadratic programming problem. Int. Trans. Oper. Res. 18(2), 257–270 (2011)

30. Mauri, G.R., Lorena, L.A.N.: A column generation approach for the unconstrained binary quadratic programming problem. Eur. J. Oper. Res. 217(1), 69–74 (2012)

31. Mauri, G.R., Lorena, L.A.N.: Improving a lagrangian decomposition for the unconstrained binary quadratic programming problem. Comput. Oper. Res. 39(7), 1577–1581 (2012)

32. Li, D., Sun, X., Liu, C.: An exact solution method for unconstrained quadratic 0–1 programming: a geometric approach. J. Glob. Optim. 52(4), 797–829 (2012)

33. Beasley, J.: Heuristic algorithms for the unconstrained binary quadratic programming problem (1998)

34. Glover, F., Kochenberger, G.A., Alidaee, B.: Adaptive memory tabu search for binary quadratic programs. Manag. Sci. 44(3), 336–345 (1998)

35. Glover, F., Kochenberger, G., Alidaee, B., Amini, M.: Meta-Heuristics. Tabu search with critical event memory: an enhanced application for binary quadratic programs, pp. 93–109. Springer, New York (1999)

36. Palubeckis, G.: Iterated tabu search for the unconstrained binary quadratic optimization problem. Informatica 17(2), 279–296 (2006)

37. Glover, F., Lü, Z., Hao, J.-K.: Diversification-driven tabu search for unconstrained binary quadratic problems. 4OR 8(3), 239–253 (2010)

38. Lü, Z., Hao, J.-K., Glover, F.: Evolutionary Computation in Combinatorial Optimization. A study of memetic search with multi-parent combination for UBQP, pp. 154–165. Springer, Heidelberg (2010)

39. Shylo, V., Shylo, O.: Systems analysis; solving unconstrained binary quadratic programming problem by global equilibrium search. Cyber. Syst. Anal. 47(6), 889–897 (2011)

40. Lü, Z., Hao, J.-K., Glover, F.: Neighborhood analysis: a case study on curriculum-based course timetabling. J. Heuristics 17(2), 97–118 (2011)

41. Wang, Y., Lü, Z., Glover, F., Hao, J.-K.: Probabilistic grasp-tabu search algorithms for the UBQP problem. Comput. Oper. Res. 40(12), 3100–3107 (2013)

42. Alkhamis, T.M., Hasan, M., Ahmed, M.A.: Simulated annealing for the unconstrained quadratic pseudo-boolean function. Eur. J. Oper. Res. 108(3), 641–652 (1998)

43. Katayama, K., Narihisa, H.: Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. Eur. J. Oper. Res. 134(1), 103–119 (2001)

44. Merz, P., Freisleben, B.: Genetic algorithms for binary quadratic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 1, pp. 417–424, Citeseer, (1999)

45. Katayama, K., Tani, M., Narihisa, H.: Solving large binary quadratic programming problems by effective genetic local search algorithm. In: GECCO, pp. 643–650 (2000)

46. Lodi, A., Allemand, K., Liebling, T.M.: An evolutionary heuristic for quadratic 0–1 programming. Eur. J. Oper. Res. 119(3), 662–670 (1999)

47. Cai, Y., Wang, J., Yin, J., Zhou, Y.: Memetic clonal selection algorithm with EDA vaccination for unconstrained binary quadratic programming problems. Expert Syst. Appl. 38(6), 7817–7827 (2011)

48. Wang, Y., Lü, Z., Glover, F., Hao, J.-K.: A multilevel algorithm for large unconstrained binary quadratic optimization. In: Integration of AI and OR Techniques in Constrain Programming for Combinatorial Optimization Problems, pp. 395–408, Springer (2012)

49. Amini, M.M., Alidaee, B., Kochenberger, G.A.: New Ideas in Optimization. A scatter search approach to unconstrained quadratic binary programs, pp. 317–330. McGraw-Hill, New York (1999)
50. Palubeckis, G., Tomkevicius, A.: [GRASP] implementations for the unconstrained binary quadratic optimization problem. Inf. Technol. Control 24, 14–20 (2002)
51. Boros, E., Hammer, P.L., Tavares, G.: Local search heuristics for quadratic unconstrained binary optimization (QUBO). J. Heuristics 13(2), 99–132 (2007)
52. Boros, E., Hammer, P.L., Tavares, G.: Preprocessing of unconstrained quadratic binary optimization. Technical Report Rutcor (2006)
53. Dantzig, G.B., Wolfe, P.: Decomposition principle for linear programs. Oper. Res. 8(1), 101–111 (1960)
54. Bertsekas, D.P.: Nonlinear Programming. Athena Scientific, Belmont (1999)
55. Boros, E., Hammer, P.: A max-flow approach to improved roof duality in quadratic 0-1 minimization. Rutgers University. Rutgers Center for Operations Research (RUTCOR) (1989)
56. Chardaire, P., Sutter, A.: A decomposition method for quadratic zero-one programming. Manag. Sci. 41(4), 704–712 (1995)
57. Boros, E., Hammer, P.L., Tavares, G.: Preprocessing of unconstrained quadratic binary optimization (2006)
58. Bian, Z., Chudak, F., Israel, R., Lackey, B., Macready, W.G., Roy, A.: Discrete optimization using quantum annealing on sparse Ising models. Interdiscip. Phys. 2, 56 (2014)
59. Zintchenko, I., Hastings, M.B., Troyer, M.: From local to global ground states in Ising spin glasses. Phys. Rev. B 91(2), 24201 (2015)
60. Kadowaki, T., Nishimori, H.: Quantum annealing in the transverse Ising model. Phys. Rev. E 58(5), 5355 (1998)
61. Finnila, A.B., Gomez, M.A., Sebenik, C., Stenson, C., Doll, J.D.: Quantum annealing: a new method for minimizing multidimensional functions. Chem. Phys. Lett. 219(5), 343–348 (1994)
62. Ray, P., Chakrabarti, B.K., Chakrabarti, A.: Sherrington-kirkpatrick model in a transverse field: absence of replica symmetry breaking due to quantum fluctuations. Phys. Rev. B 39(16), 11828 (1989)
63. Santoro, G.E., Martoňák, R., Tosatti, E., Car, R.: Theory of quantum annealing of an Ising spin glass. Science 295(5564), 2427–2430 (2002)
64. Martoňák, R., Santoro, G.E., Tosatti, E.: Quantum annealing by the path-integral monte carlo method: the two-dimensional random ising model. Phys. Rev. B 66, 094203 (2002)
65. Crosson, E., Harrow, A. W.: Simulated quantum annealing can be exponentially faster than classical simulated annealing, arXiv preprint arXiv:1601.03030 (2016)
66. Battaglia, D.A., Santoro, G.E., Tosatti, E.: Optimization by quantum annealing: lessons from hard satisfiability problems. Phys. Rev. E 71(6), 066707 (2005)
67. Lanting, T., Przybysz, A.J., Smirnov, A.Y., Spedalieri, F.M., Amin, M.H., Berkley, A.J., Harris, R., Altomare, F., Boixo, S., Bunyk, P., et al.: Entanglement in a quantum annealing processor. Phys. Rev. X 4(2), 021041 (2014)
68. Neven, H., Smelyanskiy, V. N., Boixo, S., Shabani, A., Isakov, S. V., Dykman, M., Denchev, V. S., Amin, M., Smirnov, A., Mohseni, M.: Computational role of collective tunneling in a quantum annealer. Bull. Am. Phys. Soc. 60(1) (2015)
69. King, A.D.: Performance of a quantum annealer on range-limited constraint satisfaction problems, In: Proceedings of the ACM International Conference on Computing Frontiers, ACM (2013)
70. Katzgraber, H.G., Hamze, F., Andrist, R.S.: Glassy chimeras could be blind to quantum speedup: designing better benchmarks for quantum annealing machines. Phys. Rev. X 4(2), 021008 (2014)
71. Boixo, S., Rønnow, T.F., Isakov, S.V., Wang, Z., Wecker, D., Lidar, D.A., Martinis, J.M., Troyer, M.: Evidence for quantum annealing with more than one hundred qubits. Nat. Phys. 10(3), 218–224 (2014)
72. Hen, I., Job, J., Albash, T., Rønnow, T.F., Troyer, M., Lidar, D.A.: Probing for quantum speedup in spin-glass problems with planted solutions. Phys. Rev. A 92(4), 042325 (2015)
73. King, A.D.: Performance of a quantum annealer on range-limited constraint satisfaction problems, arXiv preprint. arXiv:1502.02098 (2015)
74. Denchev, V.S., Boixo, S., Isakov, S.V., Ding, N., Babbush, R., Smelyanskiy, V., Martinis, J., Neven, H.: What is the computational value of finite range tunneling?, arXiv preprint. arXiv:1512.02206 (2015)
75. Rønnow, T.F., Wang, Z., Job, J., Boixo, S., Isakov, S.V., Wecker, D., Martinis, J.M., Lidar, D.A., Troyer, M.: Defining and detecting quantum speedup. Science 345(6195), 420–424 (2014)
76. Martin-Mayor, V., Hen, I.: Unraveling quantum annealers using classical hardness. Sci. Rep. 5 (2015)
77. Katzgraber, H.G., Hamze, F., Zhu, Z., Ochoa, A.J., Munoz-Bauza, H.: Seeking quantum speedup through spin glasses: the good, the bad, and the ugly. Phys. Rev. X 5(3), 031026 (2015)
78. Bunyk, P., Hoskinson, E.M., Johnson, M., Tolkacheva, E., Altomare, F., Berkley, A.J., Harris, R., Hilton, J.P., Lanting, T., Przybysz, A.J., Whittaker, J.: Architectural considerations in the design of a superconducting quantum annealing processor. IEEE Trans. Appl. Supercond. 24, 1–10 (2014)
79. Choi, V.: Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. Quant. Inf. Process. 7(5), 193–209 (2008)
80. Choi, V.: Minor-embedding in adiabatic quantum computation: II. Minor-universal graph design. Quant. Inf. Process. 10(3), 343–353 (2011)
81. Williams, C.: State-of-the-art quantum annealing and its application to cryptology, Isaac Newton Institute (2014). http://sms.cam.ac.uk/media/1804114
82. Merz, P., Katayama, K.: Memetic algorithms for the unconstrained binary quadratic programming problem. BioSystems 78(1), 99–118 (2004)
83. Glover, F., Hao, J.-K.: Efficient evaluations for solving large 0–1 unconstrained quadratic optimisation problems. Int. J. Metaheuristics 1(1), 3–10 (2010)
84. Wang, Y., Lü, Z., Glover, F., Hao, J.-K.: Path relinking for unconstrained binary quadratic programming. Eur. J. Oper. Res. 223(3), 595–604 (2012)
85. Beasley, J.E.: OR-Library: Unconstrained binary quadratic programming,” (2014). http://people.brunel.ac.uk/~mastjjb/jeb/orlib/bqpinfo.html
86. Tavares, G.: New algorithms for Quadratic Unconstrained Binary Optimization (QUBO) with applications in engineering and social sciences. PhD thesis, Rutgers University, Graduate School - New Brunswick (2008)
87. Palubeckis, G.: Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. Ann. Oper. Res. 131(1–4), 259–282 (2004)
88. Puden, K.L., Albash, T., Lidar, D.A.: Error-corrected quantum annealing with hundreds of qubits. Nat. Commun. 5 (2014)
89. Puden, K.L., Albash, T., Lidar, D.A.: Quantum annealing correction for random ising problems. Phys. Rev. A 91, 042302 (2015)
90. Barends, R., Lamata, L., Kelly, J., Garcia-Alvarez, L., Fowler, A.G., Megrant, A., Jeffrey, E., White, T.C., Sank, D., Mutus, J.Y., Campbell, B., Chen, Y., Chen, Z., Chiaro, B., Dunsworth, A., Hoi, I.C., Neill, C., O’Malley, P.J.J., Quintana, C., Roushan, P., Vainsencher, A., Wenner, J., Solano, E., Martinis, J.M.: Digital quantum simulation of fermionic models with a superconducting circuit. Nat. Commun. 6, 07 (2015)
91. Lechner, W., Hauke, P., Zoller, P.: A quantum annealing architecture with all-to-all connectivity from local interactions. Sci. Adv. 1(9), e1500838 (2015)
92. Pastawski, F., Preskill, J.: Error correction for a proposed quantum annealing architecture, arXiv preprint. arXiv:1511.00004 (2015)