1QBit

Building an Iterative Heuristic Solver for a Quantum Annealer

Gili Rosenberg, Mohammad Vazifeh, Brad Woods, Eldad Haber

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 to 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.

Keywords: Quantum annealing, Quadratic unconstrained binary optimization, Combinatoric optimization, k-opt, Local search, Iterative heuristic solver

1 Introduction

1.1 The problem and previous work

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

$$\min \quad x^T Q x$$
$$s.t. \quad x \in S,$$

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 [18], Boros and Hammer [15], Bourjolly et al. [19], Du and Pardalos [26], Pardalos and Rodgers [67,69], Pardalos and Xue [70], and Kochenberger et al. [47]).

Numerous interesting applications expressed naturally in the form of a QUBO problem have appeared in the literature. Barahona et al. [6,25] formulate and solve the problem of finding exact ground states of spin glasses with magnetic fields. Alidaee et al. [2] 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. [14] 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. The QUBO problem has been used in the prediction of epileptic seizures [40]. Alidaee et al. [1] discuss the number partitioning problem as a QUBO problem.

To solve a QUBO problem, a number of exact methods have been developed [5,12,22,34–37,39,49,55–57,62,66–68,71,72,84]. 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 [10,29,31,33,51,52,65,78,82], simulated annealing [3,42], evolutionary algorithms [21,43,50,51,59,80], scatter search [4], GRASP [63,82], and iterated local search algorithms [17] have been proposed. Additionally, preprocessing techniques have been developed to simplify the solving of QUBO problems [16].

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 Section 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.

While working on this research we noted research on the possibility of solving larger problems by decomposing them into smaller problems and solving them on the D-Wave machine [11]. 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 [85]. The authors of that work explored the applicability of solution via local updates to two- and three-dimensional Ising spin glasses, as well as Chimera-structured problems. 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[1], 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 [27, 41, 75]. Indeed, simulated quantum annealing [54, 77] shows a speedup for some problem classes [77], but not for some others [8].

There is strong evidence that the D-Wave machine is quantum [48, 61], but it is yet to be established whether it provides a quantum speedup over classical computers. Recently, there has been significant interest in benchmarking the D-Wave machines [13, 38, 44, 46, 58], and there is ongoing debate on how to define quantum speedup, and on which problems a quantum annealer could be expected to demonstrate it [45, 53, 76]. We expect results in the near future to shed light on this important point.

The connectivity of the D-Wave processor's qubits is currently given by the Chimera graph [20]. Due to the sparsity of this hardware graph, it is often necessary to identify multiple physical qubits with one logical qubit in order to increase the connectivity. In particular, it is known that the largest complete graph that can be embedded on a Chimera graph of size n is of size $\sqrt{2n} + 1$ (for instance, for the current-generation 512-qubit chip[2], the largest complete graph is of size 33) [23, 24]. The parameter misspecification error rate for the current-generation chip is 3–5%, whereas the next generation of chip, scheduled for release in the second half of 2015, will have 1152 active qubits and the error rate reduced by an expected 33% [83].

The remainder of this paper is organized as follows. In Section 2 we describe the algorithm. In Section 3 we explain the benchmarking methodology and present computational results. In Sections 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 Section 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).

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 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

---

[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.

[2] The chip is current as of mid-July 2015.

Algorithm 1 Basic Algorithm

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

---

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 (Section 2.3)

2. Different underlying optimizers (Section 2.4)

3. Escape strategies (Section 2.5)

4. Choice of variables (Section 2.6)

5. Tabu search for k-opt (Section 2.7)

6. 1-opt tabu search improvement (Section 2.8)

7. Stopping criteria and convergence criterion (Section 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[3] is that once they are initialized, they can be updated very efficiently [28, 60], 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 Section 2.6.1).

2.3 Better initial guesses

We use the greedy heuristic given by Merz and Katayama [60] to find good (i.e. 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, we experimented with using a deterministic version of this greedy heuristic for the first element in the reference 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).

---

[3]A one-flip gain is the change in the objective function's value given a flip of a single bit. We refer to the collection of all possible single flips and their corresponding change in the objection function's value as the "one-flip gains".

Algorithm 2 Improved Algorithm

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

We note that the randomized version by Merz and Katayama [60] 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 II 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 $10k$, 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.

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", 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. [81], 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

Algorithm 3 Randomized Greedy Initial Guess

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

Algorithm 4 Deterministic Greedy Initial Guess

1: procedure DeterministicGreedy
2: $\quad C \leftarrow \{1, ..., N\}$
3: $\quad$ for $i \in \{1, ..., N\}$ do $x_i = \frac{1}{2}$
4: $\quad$ Calculate one-flip gains g_i for all $i \in \{1, ..., N\}$
5: $\quad$ repeat
6: $\quad\quad$ Find k_0 with $g_k^0 = \min_{i \in C} g_i^0$ and k_1 with $g_k^1 = \min_{i \in C} g_i^1$
7: $\quad\quad$ if $g_{k_0}^0 < g_{k_1}^1$ then
8: $\quad\quad\quad x_{k_0} \leftarrow 0;\ C \leftarrow C - \{k_0\}$
9: $\quad\quad$ else
10: $\quad\quad\quad x_{k_1} \leftarrow 1;\ C \leftarrow C - \{k_1\}$
11: $\quad\quad$ Update one-flip gains g_i for all $i \in C$;
12: $\quad$ until $C = \{\}$
13: $\quad$ return x

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

1: procedure PathRelinkingEscape
2:    if $x \notin$ 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:    if ref_set is full then
12:        if unfused_pairs is not empty then
13:            $x \leftarrow$ Fuse random pair from unfused_pairs
14:        else
15:            Empty ref_set except best
16:            Empty unfused_pairs
17:            $x \leftarrow$ RandomizedGreedy()
18:    else $x \leftarrow$ RandomizedGreedy()
19:    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 $V_{\min}$, 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. [30, 32].

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 Sections 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 Section 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 Section 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

1: procedure CouplingChoice
2:      indices_to_update ← {} (an empty set)
3:      $LC \leftarrow C$
4:      $LQ \leftarrow$ project Q on C
5:      $n \leftarrow \dim(LQ)$
6:      for $i = 1$ to n do strength[i] $\leftarrow \sum_j |LQ|_{ji}$
7:      probabilities[i] $\leftarrow$ strength[i] / sum(strength), for each i
8:      while length(indices_to_update) $< k$ do
9:          $p \leftarrow$ random$(0, 1)$ (first stage)
10:         $i \leftarrow C$[index for interval in which p falls in cumulative probabilities]
11:         if $i \in LC$ 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 $j \in LC$ do conditional_strengths[j] $\leftarrow |LQ_{ji}|$
15:         if $\sum$(conditional_strengths) $= 0$ then continue
16:         Divide conditional_probabilities by $\sum$(conditional_strengths)
17:         $p \leftarrow$ random$(0, 1)$ (second stage)
18:         $m \leftarrow LC$[index for interval in which p 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. 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 often referred to as an "embedding" [23, 24].

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 $k \leq \sqrt{2n} + 1$, 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.

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 ~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 k-opt). 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, [30]), 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 seconds[4]). 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 Section 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 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 seconds of wall-clock time for the problems of size 500, 2250 seconds for the problems of size 2500, and 1.2*N* seconds 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 [9]. We generated these problems (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 [79].

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. [81]. The third set of problems was generated using the problem generator and settings written by Palubeckis[5] [64] with densities ranging from 0.5 to 1, and we used the best known solutions reported in Wang et al. [81].

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

---

[4]We chose 0.02 seconds based on 1000 anneals each taking 20 microseconds (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.

[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.

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 as well as prior literature (such as [81]). The names and descriptions of the parameters that were varied are in Table 2.

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 underlying 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 sec	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 $\propto N/k$
W	tabu whole group (True/False)

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 Section 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 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 Table 4 and Table 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

Table 3: Results for size 500 for $k = 50$, $CL = 3$, $TT = 64$, $w = 1$, fusion-guided path relinking with $W =$ True. The average time to best solution in seconds (assuming an underlying optimizer call time of 0.02 seconds) is denoted $<T>$, the average gap to the best known solution as a percentage is denoted $<G>$, 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 $<I>$.

problem set	$<T>$	STD(T)	$<G>$	STD(G)	succ.	$<I>$	STD(I)
bqp500	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

Table 4: Detailed results for OR-Library 500 for $k = 50$, $CL = 3$, $TT = 6$, $w = 1$, fusion-guided path relinking with $W =$ True. See the caption of Table 3 for a definition of $<T>$, $<G>$, "succ.", and $<I>$.

name	$<T>$	STD(T)	$<G>$	STD(G)	succ.	$<I>$	STD(I)
bqp500-1	5.85	2.51	0.10	0.13	50.00	197.3	87.9
bqp500-2	3.74	2.54	0.00	0.01	90.62	118.3	88.3
bqp500-3	1.55	1.00	0.00	0.00	100.00	45.8	33.7
bqp500-4	5.93	1.89	0.01	0.01	34.38	188.2	67.1
bqp500-5	3.52	1.82	0.00	0.02	96.88	113.7	64.1
bqp500-6	6.25	1.80	0.04	0.04	46.88	206.3	64.2
bqp500-7	6.35	2.10	0.02	0.02	37.50	209.4	75.6
bqp500-8	6.34	2.10	0.04	0.02	12.50	211.5	76.7
bqp500-9	5.96	1.99	0.02	0.04	40.62	204.9	71.1
bqp500-10	2.85	1.75	0.01	0.06	96.88	87.2	58.7

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.

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 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 with 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 training data, because it will not generalize to the next set of data, and for this reason optimizers will often intentionally be stopped early.

We investigated how the time required varied as we change the desired gap. Fig. 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

Table 5: Detailed results for RFDQ 500 for $k = 50$, $CL = 3$, $TT = 8$, $w = 1$, fusion-guided path relinking with $W =$ True. See the caption of Table 3 for a definition of $<T>$, $<G>$, "succ.", and $<I>$.

name	$<T>$	STD(T)	$<G>$	STD(G)	succ.	$<I>$	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

Table 6: Results for size 2500 for $k = 50$, $CL = 3$, $TT = 30$, $w = 1$, fusion-guided path relinking with $W =$ True. See the caption of Table 3 for a definition of $<T>$, $<G>$, "succ.", and $<I>$.

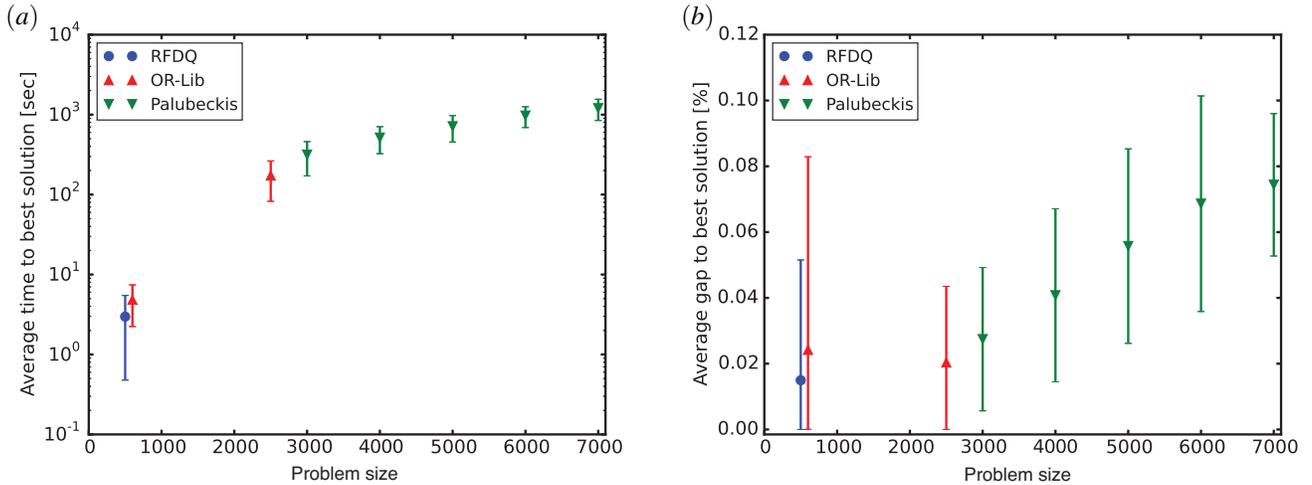| identifier	$<T>$	STD(T)	$<G>$	STD(G)	succ.	$<I>$	STD(I)
bqp2500	173.07	90.76	0.02	0.02	34.06	3798.0	2008.6

Figure 1: (a) Average time as a function of the problem size. Data was collected for GQS with parameters $k = 50$, $CL = 3$, $w = 1$, $TT = 0.6N/k$, fusion-guided path relinking with $W =$ True, averaged over the respective problems and 32 repetitions per problem. (b) Average gap as a function of problem size. Note for both figures: the results for OR-Library 500 were moved slightly to the right for clarity.

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. 3(a). We found that the time to best solution falls exponentially, 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. 3(b). The expectation is that as $k \to N$, 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 seconds). 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

Table 7: Detailed results for RFDQ 500 for $k = 50$, $CL = 3$, $TT = 30$, $w = 1$, fusion-guided path relinking with $W = $ True. See the caption of Table 3 for a definition of $<T>$, $<G>$, "succ.", and $<I>$.

| name	$<T>$	STD(T)	$<G>$	STD(G)	succ.	$<I>$	STD(I)
bqp2500-1	180.63	82.01	0.01	0.01	34.38	3966.6	1804.7
bqp2500-2	197.50	95.32	0.05	0.02	0.00	4381.8	2091.1
bqp2500-3	190.15	74.70	0.03	0.02	9.38	4240.1	1679.0
bqp2500-4	77.32	49.22	0.00	0.00	100.00	1634.7	1070.9
bqp2500-5	140.69	87.81	0.00	0.00	62.50	3033.3	1915.4
bqp2500-6	198.85	80.40	0.01	0.01	25.00	4334.7	1761.6
bqp2500-7	245.32	67.62	0.03	0.02	18.75	5387.8	1472.5
bqp2500-8	170.00	85.64	0.01	0.01	46.88	3716.1	1866.9
bqp2500-9	148.07	83.10	0.01	0.01	43.75	3241.9	1854.1
bqp2500-10	182.20	88.32	0.05	0.02	0.00	4043.4	1965.8
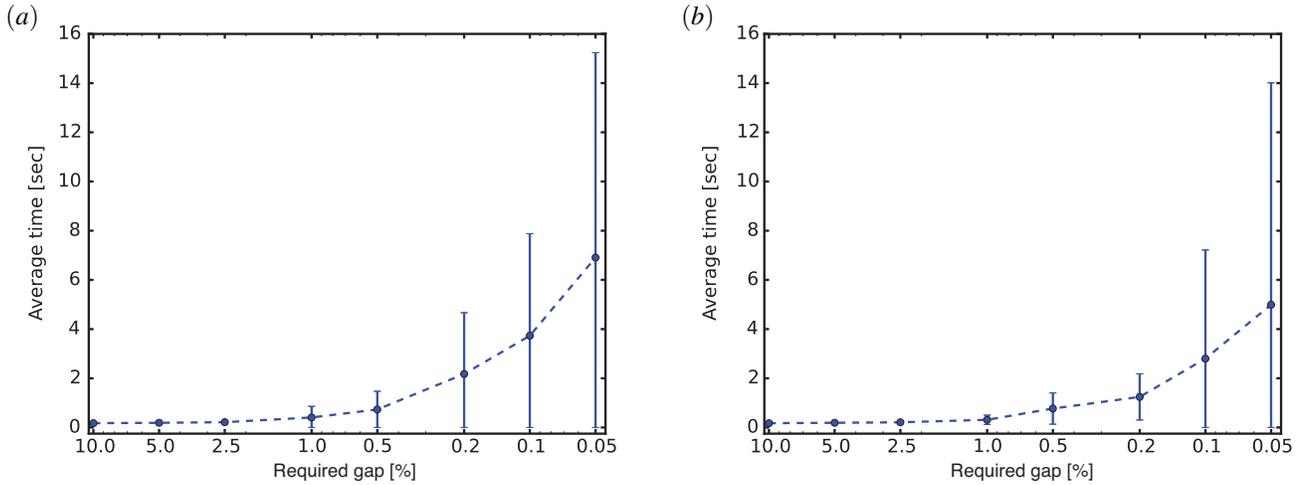
Figure 2: (a) Average time as a function of the required gap. Data was collected for GQS with parameters $k = 50$, $CL = 3$, $TT = 8$, $w = 1$, fusion-guided path relinking with $W = $ True, for OR-Library problems of size 500, averaged over the ten problems and 32 repetitions. (b) The same for RFDQ problems of size 500.

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.

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

Table 8: Detailed results for Palubeckis instances of size 3000–7000 for $k = 50$, $CL = 3$, $w = 1$, fusion-guided path relinking with $W = \text{True}$. See the caption of Table 3 for a definition of $<T>$, $<G>$, "succ.", and $<I>$.

| 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	2481.5
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	0.62	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
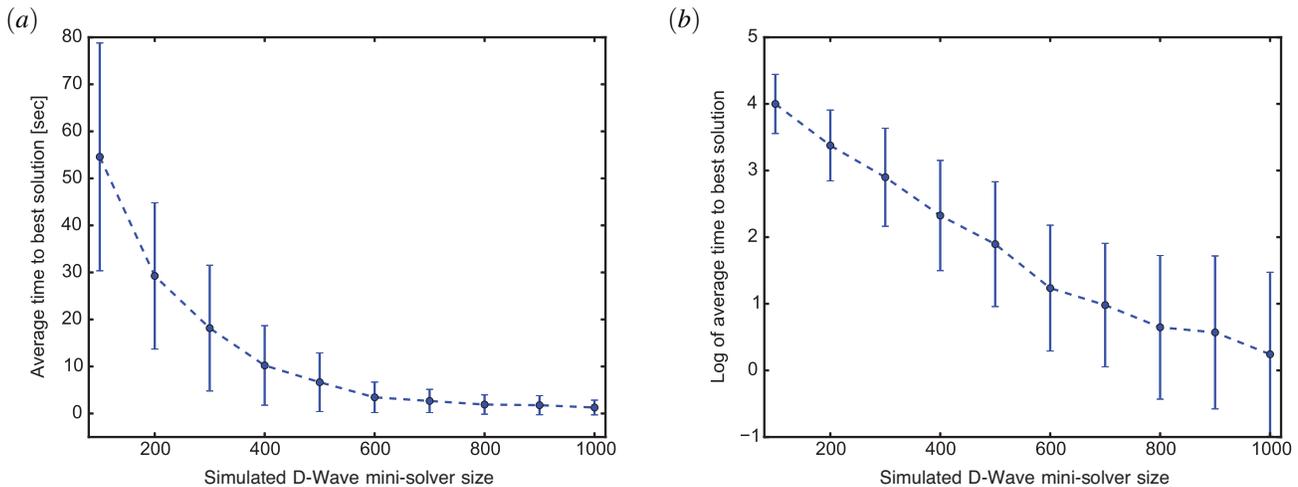
Figure 3: (a) Average time to best solution as a function of k, the simulated D-Wave chip size. Data was collected for GQS with parameters $CL = 3$, $w = 1$, fusion-guided path relinking with $W = \text{True}$ for OR-Library problems of size 2500, averaged over the ten problems and 32 repetitions, and 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)).

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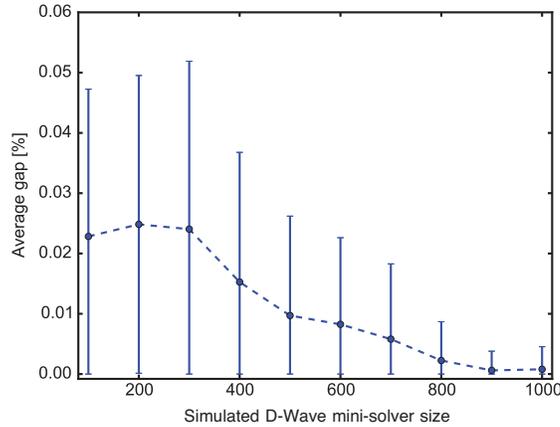
Figure 4: Average gap as a function of the simulated D-Wave chip size, with the same parameters as in Fig. 3.

Table 9: 1-opt tabu results. See the caption of Table 3 for a definition of $<T>$, $<G>$, and "succ". The average number of one-bit flips is denoted $<I>$.

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

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 Section 3.3.5). Exactly when this occurs will depend on the quality and run time of the D-Wave processor.

5 Conclusions and Future Work

We have shown that it is possible to use the D-Wave machine to heuristically solve significantly larger problems than the chip intrinsically allows. If in the future the 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 problems 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 seconds, solving these problems requires 3–5 seconds. Keeping the solver size constant, for larger problem sizes, the number of iterations increases rapidly and the quality of the produced 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 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. Similarly, future advances in error correction [7, 73, 74] 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 estimated.

One way in which our algorithm could be improved is by implementing preprocessing. There are many ways in which QUBO problems can be preprocessed to decrease the size of the problem (several are reviewed in [79]). In addition, using a better underlying optimizer such as the path relinking algorithm would undoubtedly improve the results [81]. 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 Section 2.6.4).

Acknowledgements

References

[1] Bahram Alidaee, Fred Glover, Gary A. Kochenberger, and Cesar Rego. A new modeling and solution approach for the number partitioning problem. J. Appl. Math. Decis. Sci., 2005(2):113–121, 2005.

[2] Bahram Alidaee, Gary A. Kochenberger, and Ahmad Ahmadian. 0-1 quadratic programming approach for optimum solutions of two scheduling problems. Internat. J. Systems Sci., 25(2):401–408, 1994.

[3] Talal M. Alkhamis, Merza Hasan, and Mohamed A. Ahmed. Simulated annealing for the unconstrained quadratic pseudo-boolean function. European Journal of Operational Research, 108(3):641 – 652, 1998.

[4] Mohammad M Amini, Bahram Alidaee, and Gary A Kochenberger. A scatter search approach to unconstrained quadratic binary programs. In New ideas in optimization, pages 317–330. McGraw-Hill Ltd., UK, 1999.

[5] F. Barahona, M. Jünger, and G. Reinelt. Experiments in quadratic 0–1 programming. Mathematical Programming, 44(1-3):127–137, 1989.

[6] Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. Operations Research, 36(3):493–513, 1988.

[7] R. Barends, L. Lamata, J. Kelly, L. Garcia-Alvarez, A. G. Fowler, A Megrant, E Jeffrey, T. C. White, D. Sank, J. Y. Mutus, B. Campbell, Yu Chen, Z. Chen, B. Chiaro, A. Dunsworth, I. C. Hoi, C. Neill, P. J. J. O/'Malley, C. Quintana, P. Roushan, A. Vainsencher, J. Wenner, E. Solano, and John M. Martinis. Digital quantum simulation of fermionic models with a superconducting circuit. Nat Commun, 6, 07 2015.

[8] Demian A. Battaglia, Giuseppe E. Santoro, and Erio Tosatti. Optimization by quantum annealing: Lessons from hard satisfiability problems. Physical Review E, 71(6):066707, 2005.

[9] J. E. Beasley. OR-Library: Unconstrained binary quadratic programming, December 2014. http://people.brunel.ac.uk/ mastjjb/jeb/orlib/bqpinfo.html.

[10] J.E. Beasley. Heuristic algorithms for the unconstrained binary quadratic programming problem, 1998.

[11] Zhengbing Bian, Fabian Chudak, Robert Israel, Brad Lackey, William G. Macready, and Aidan Roy. Discrete optimization using quantum annealing on sparse Ising models. Interdisciplinary Physics, 2:56, 2014.

[12] A. Billionnet and A. Sutter. Minimization of a quadratic pseudo-boolean function. European Journal of Operational Research, 78(1):106 – 115, 1994.

[13] Sergio Boixo, Troels F. Rønnow, Sergei V. Isakov, Zhihui Wang, David Wecker, Daniel A. Lidar, John M. Martinis, and Matthias Troyer. Evidence for quantum annealing with more than one hundred qubits. Nature Physics, 10(3):218–224, 2014.

[14] Immanuel M Bomze, Marco Budinich, Panos M Pardalos, and Marcello Pelillo. The maximum clique problem. In Handbook of combinatorial optimization, pages 1–74. Springer, 1999.

[15] Endre Boros and Peter L. Hammer. Pseudo-Boolean optimization. Discrete Appl. Math., 123(1-3):155–225, 2002. Workshop on Discrete Optimization, DO'99 (Piscataway, NJ).

[16] Endre Boros, Peter L Hammer, and Gabriel Tavares. Preprocessing of unconstrained quadratic binary optimization. Technical report, Rutcor, 2006.

[17] Endre Boros, Peter L. Hammer, and Gabriel Tavares. Local search heuristics for quadratic unconstrained binary optimization (QUBO). Journal of Heuristics, 13(2):99–132, 2007.

[18] Endre Boros and András Prékopa. Probabilistic bounds and algorithms for the maximum satisfiability problem. Ann. Oper. Res., 21(1-4):109–126, 1989.

[19] Jean-Marie Bourjolly. A quadratic 0-1 optimization algorithm for the maximum clique and stable set problems. Technical report, Univ. of Michigan, Ann Arbor, MI (United States), 1994.

[20] P. Bunyk, E. M. Hoskinson, M. Johnson, Elena Tolkacheva, Fabio Altomare, A. J. Berkley, Richard Harris, J. P. Hilton, Trevor Lanting, A. J. Przybysz, and J. Whittaker. Architectural considerations in the design of a superconducting quantum annealing processor. IEEE Transactions on Applied Superconductivity, 24, 2014.

[21] Yiqiao Cai, Jiahai Wang, Jian Yin, and Yalan Zhou. Memetic clonal selection algorithm with {EDA} vaccination for unconstrained binary quadratic programming problems. Expert Systems with Applications, 38(6):7817 – 7827, 2011.

[22] Michael W. Carter. The indefinite zero-one quadratic problem. Discrete Applied Mathematics, 7(1):23 – 44, 1984.

[23] Vicky Choi. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. Quantum Information Processing, 7(5):193–209, 2008.

[24] Vicky Choi. Minor-embedding in adiabatic quantum computation: II. minor-universal graph design. Quantum Information Processing, 10(3):343–353, 2011.

[25] Caterina De Simone, Martin Diehl, Michael Jünger, Petra Mutzel, Gerhard Reinelt, and Giovanni Rinaldi. Exact ground states of ising spin glasses: New experimental results with a branch-and-cut algorithm. Journal of Statistical Physics, 80(1-2):487–496, 1995.

[26] Ding-Zhu Du and Panos M Pardalos. Handbook of combinatorial optimization: supplement, volume 1. Springer Science & Business Media, 1999.

[27] A. B. Finnila, M. A. Gomez, C. Sebenik, C. Stenson, and J. D. Doll. Quantum annealing: A new method for minimizing multidimensional functions. Chemical physics letters, 219(5):343–348, 1994.

[28] Fred Glover and Jin-Kao Hao. Efficient evaluations for solving large 0-1 unconstrained quadratic optimisation problems. International Journal of Metaheuristics, 1(1):3–10, 2010.

[29] Fred Glover, Gary Kochenberger, Bahram Alidaee, and Mohammed Amini. Tabu search with critical event memory: an enhanced application for binary quadratic programs. In Meta-Heuristics, pages 93–109. Springer, 1999.

[30] Fred Glover, Gary Kochenberger, Bahram Alidaee, and Mohammed Amini. Tabu search with critical event memory: an enhanced application for binary quadratic programs. In Meta-Heuristics, pages 93–109. Springer, 1999.

[31] Fred Glover, Gary A. Kochenberger, and Bahram Alidaee. Adaptive memory tabu search for binary quadratic programs. Management Science, 44(3):336–345, 1998.

[32] Fred Glover, Gary A. Kochenberger, and Bahram Alidaee. Adaptive memory tabu search for binary quadratic programs. Management Science, 44(3):336–345, 1998.

[33] Fred Glover, Zhipeng Lü, and Jin-Kao Hao. Diversification-driven tabu search for unconstrained binary quadratic problems. 4OR, 8(3):239–253, 2010.

[34] Serigne Gueye and Philippe Michelon. A linearization framework for unconstrained quadratic (0-1) problems. Discrete Applied Mathematics, 157(6):1255 – 1266, 2009. Reformulation Techniques and Mathematical Programming.

[35] V.P. Gulati, S.K. Gupta, and A.K. Mittal. Unconstrained quadratic bivalent programming problem. European Journal of Operational Research, 15(1):121 – 125, 1984.

[36] Pierre Hansen, Brigitte Jaumard, Christophe Meyer, and Québec) Groupe 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.

[37] Christoph Helmberg and Franz Rendl. Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. Mathematical Programming, 82(3):291–315, 1998.

[38] Itay Hen, Joshua Job, Tameem Albash, Troels F Rønnow, Matthias Troyer, and Daniel A Lidar. Probing for quantum speedup in spin-glass problems with planted solutions. Physical Review A, 92(4):042325, 2015.

[39] Hong-Xuan Huang, PanosM. Pardalos, and OlegA. Prokopyev. Lower bound improvement and forcing rule for quadratic binary programming. Computational Optimization and Applications, 33(2-3):187–208, 2006.

[40] L. D. Iasemidis, P. Pardalos, J. C. Sackellares, and D.-S. Shiau. Quadratic binary programming and dynamical system approach to determine the predictability of epileptic seizures. J. Comb. Optim., 5(1):9–26, 2001.

[41] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse Ising model. Physical Review E, 58(5):5355, 1998.

[42] Kengo Katayama and Hiroyuki Narihisa. Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. European Journal of Operational Research, 134(1):103 – 119, 2001.

[43] Kengo Katayama, Masafumi Tani, and Hiroyuki Narihisa. Solving large binary quadratic programming problems by effective genetic local search algorithm. In GECCO, pages 643–650, 2000.

[44] Helmut G. Katzgraber, Firas Hamze, and Ruben S. Andrist. Glassy chimeras could be blind to quantum speedup: Designing better benchmarks for quantum annealing machines. Physical Review X, 4(2):021008, 2014.

[45] Helmut G Katzgraber, Firas Hamze, Zheng Zhu, Andrew J Ochoa, and H Munoz-Bauza. Seeking quantum speedup through spin glasses: The good, the bad, and the ugly. Physical Review X, 5(3):031026, 2015.

[46] Andrew D. King. Performance of a quantum annealer on range-limited constraint satisfaction problems. arXiv preprint arXiv:1502.02098, 2015.

[47] Gary A Kochenberger, Fred Glover, Bahram Alidaee, and Cesar Rego. A unified modeling and solution framework for combinatorial optimization problems. OR Spectrum, 26(2):237–250, 2004.

[48] T. Lanting, A. J. Przybysz, A. Yu Smirnov, F. M. Spedalieri, M. H. Amin, A. J. Berkley, R. Harris, F. Altomare, S. Boixo, P. Bunyk, et al. Entanglement in a quantum annealing processor. Physical Review X, 4(2):021041, 2014.

[49] D. Li, X.L. Sun, and C.L. Liu. An exact solution method for unconstrained quadratic 0–1 programming: a geometric approach. Journal of Global Optimization, 52(4):797–829, 2012.

[50] Andrea Lodi, Kim Allemand, and Thomas M. Liebling. An evolutionary heuristic for quadratic 0–1 programming. European Journal of Operational Research, 119(3):662 – 670, 1999.

[51] Zhipeng Lü, Jin-Kao Hao, and Fred Glover. A study of memetic search with multi-parent combination for UBQP. In Evolutionary Computation in Combinatorial Optimization, pages 154–165. Springer, 2010.

[52] Zhipeng Lü, Jin-Kao Hao, and Fred Glover. Neighborhood analysis: a case study on curriculum-based course timetabling. Journal of Heuristics, 17(2):97–118, 2011.

[53] Victor Martin-Mayor and Itay Hen. Unraveling quantum annealers using classical hardness. Scientific reports, 5, 2015.

[54] Roman Martoňák, Giuseppe E. Santoro, and Erio Tosatti. Quantum annealing by the path-integral monte carlo method: The two-dimensional random ising model. Phys. Rev. B, 66:094203, Sep 2002.

[55] Geraldo Regis Mauri and Luiz Antonio Nogueira Lorena. Lagrangean decompositions for the unconstrained binary quadratic programming problem. International Transactions in Operational Research, 18(2):257–270, 2011.

[56] Geraldo Regis Mauri and Luiz Antonio Nogueira Lorena. A column generation approach for the unconstrained binary quadratic programming problem. European Journal of Operational Research, 217(1):69 – 74, 2012.

[57] Geraldo Regis Mauri and Luiz Antonio Nogueira Lorena. Improving a lagrangian decomposition for the unconstrained binary quadratic programming problem. Computers and Operations Research, 39(7):1577 – 1581, 2012.

[58] Catherine C. McGeoch and Cong Wang. Experimental evaluation of an adiabatic quantum system for combinatorial optimization. In Proceedings of the ACM International Conference on Computing Frontiers. ACM, 2013.

[59] Peter Merz and Bernd Freisleben. Genetic algorithms for binary quadratic programming. In Proceedings of the genetic and evolutionary computation conference, volume 1, pages 417–424. Citeseer, 1999.

[60] Peter Merz and Kengo Katayama. Memetic algorithms for the unconstrained binary quadratic programming problem. BioSystems, 78(1):99–118, 2004.

[61] Hartmut Neven, Vadim N. Smelyanskiy, Sergio Boixo, Alireza Shabani, Sergei V. Isakov, Mark Dykman, Vasil S. Denchev, Mohammad Amin, Anatoly Smirnov, and Masoud Mohseni. Computational role of collective tunneling in a quantum annealer. Bulletin of the American Physical Society, 60(1), 2015.

[62] G. Palubeckis. A heuristic-based branch and bound algorithm for unconstrained quadratic zero-one programming. Computing, 54(4):283–301, 1995.

[63] G Palubeckis and A Tomkevicius. {GRASP} implementations for the unconstrained binary quadratic optimization problem. Information Technology and Control, 24:14–20, 2002.

[64] Gintaras Palubeckis. Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. Annals of Operations Research, 131(1-4):259–282, 2004.

[65] Gintaras Palubeckis. Iterated tabu search for the unconstrained binary quadratic optimization problem. Informatica, 17(2):279–296, 2006.

[66] Shaohua Pan, Tao Tan, and Yuxi Jiang. A global continuation algorithm for solving binary quadratic programming problems. Computational Optimization and Applications, 41(3):349–362, 2008.

[67] P. M. Pardalos and G. P. Rodgers. Computational aspects of a branch and bound algorithm for quadratic zero-one programming. Computing, 45(2):131–144, 1990.

[68] Panos M Pardalos, Oleg A Prokopyev, and Stanislav Busygin. Continuous approaches for solving discrete optimization problems. In Handbook on modelling for discrete optimization, pages 39–60. Springer, 2006.

[69] Panos M Pardalos and Gregory P Rodgers. A branch and bound algorithm for the maximum clique problem. Computers & operations research, 19(5):363–375, 1992.

[70] Panos M. Pardalos and Jue Xue. The maximum clique problem. J. Global Optim., 4(3):301–328, 1994.

[71] P.M. Pardalos and Rodgers. Parallel branch and bound algorithms for quadratic zero–one programs on the hypercube architecture. Annals of Operations Research, 22(1):271–292, 1990.

[72] Tao Pham Dinh, Nam Nguyen Canh, and HoaiAn Le Thi. An efficient combined dca and bnb using dc/sdp relaxation for globally solving binary quadratic programs. Journal of Global Optimization, 48(4):595–632, 2010.

[73] Kristen L. Pudenz, Tameem Albash, and Daniel A. Lidar. Error-corrected quantum annealing with hundreds of qubits. Nature communications, 5, 2014.

[74] Kristen L. Pudenz, Tameem Albash, and Daniel A. Lidar. Quantum annealing correction for random ising problems. Phys. Rev. A, 91:042302, Apr 2015.

[75] Pulak Ray, Bikas K. Chakrabarti, and Arunava Chakrabarti. Sherrington-kirkpatrick model in a transverse field: Absence of replica symmetry breaking due to quantum fluctuations. Physical Review B, 39(16):11828, 1989.

[76] Troels F Rønnow, Zhihui Wang, Joshua Job, Sergio Boixo, Sergei V Isakov, David Wecker, John M Martinis, Daniel A Lidar, and Matthias Troyer. Defining and detecting quantum speedup. Science, 345(6195):420–424, 2014.

[77] Giuseppe E. Santoro, Roman Martoňák, Erio Tosatti, and Roberto Car. Theory of quantum annealing of an Ising spin glass. Science, 295(5564):2427–2430, 2002.

[78] V.P. Shylo and O.V. Shylo. Systems analysis; solving unconstrained binary quadratic programming problem by global equilibrium search. Cybernetics and Systems Analysis, 47(6):889–897, 2011.

[79] Gabriel Tavares. New algorithms for Quadratic Unconstrained Binary Optimization (QUBO) with applications in engineering and social sciences. PhD thesis, Rutgers University, Graduate School - New Brunswick, 2008.

[80] Yang Wang, Zhipeng Lü, Fred Glover, and Jin-Kao Hao. A multilevel algorithm for large unconstrained binary quadratic optimization. In Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems, pages 395–408. Springer, 2012.

[81] Yang Wang, Zhipeng Lü, Fred Glover, and Jin-Kao Hao. Path relinking for unconstrained binary quadratic programming. European Journal of Operational Research, 223(3):595–604, 2012.

[82] Yang Wang, Zhipeng Lü, Fred Glover, and Jin-Kao Hao. Probabilistic grasp-tabu search algorithms for the UBQP problem. Computers and Operations Research, 40(12):3100 – 3107, 2013.

[83] Colin Williams. State-of-the-art quantum annealing and its application to cryptology. Isaac Newton Institute, 2014. http://sms.cam.ac.uk/media/1804114.

[84] H Paul Williams. Model building in linear and integer programming. Springer, 1985.

[85] Ilia Zintchenko, Matthew B Hastings, and Matthias Troyer. From local to global ground states in ising spin glasses. Physical Review B, 91(2):24201, 2015.

White Paper Summary -
Academic Paper -
Demonstration Software - http://demo.1qbit.com/

Visit 1QBit.com for more information.

1QBit