The Knowledge Engineering Review, Vol. 31:5, 465–474.
doi:10.1017/S0269888916000278

Comparing planning problem compilation approaches for quantum annealing

BRYAN O'GORMAN[1,4], ELEANOR GILBERT RIEFFEL[1], MINH DO[2,4], DAVIDE VENTURELLI[1,3] and JEREMY FRANK[2]

[1]*QuAIL, NASA Ames Research Center, Moffett Field, CA, USA;*
e-mail: bryan.a.ogorman@nasa.gov, eleanor.rieffel@nasa.gov;
[2]*Intelligent Systems Division, NASA Ames Research Center, Moffett Field, CA, USA;*
e-mail: jeremy.d.frank@nasa.gov;
[3]*Universities Space Research Association 615 National Avenue, Suite 220 Mountain View, CA 94043, USA;*
e-mail: davide.venturelli@nasa.gov;
[4]*Stinger Ghaffarian Technologies, Inc. 7701 Greenbelt Road, Suite 400 Greenbelt, MD 20770, USA;*
e-mail: minh.do@nasa.gov

Abstract

One approach to solving planning problems is to compile them to other problems for which powerful off-the-shelf solvers are available; common targets include SAT, CSP, and MILP. Recently, a novel optimization technique has become available: quantum annealing (QA). QA takes as input problem instances of quadratic unconstrained binary optimization (QUBO) problem. Early quantum annealers are now available, though their constraints restrict the types of QUBOs they can take as input. Here, we introduce the planning community to the key steps in compiling planning problems to QA hardware: a hardware-independent step, mapping, and a hardware-dependent step, embedding. After describing two approaches to mapping general planning problems to QUBO, we describe preliminary results from running an early quantum annealer on a parametrized family of hard planning problems. The results show that different mappings can substantially affect performance, even when many features of the resulting instances are similar. We conclude with insights gained from this early study that suggest directions for future work.

1 Introduction

One approach to solving planning problems is to compile them to another problem for which powerful off-the-shelf solvers are available; common targets include Satisfiability (SAT), Constraint Satisfaction Problems (CSPs), and Mixed Integer Linear Programming (MILP). Recently, a novel optimization technique has become available: quantum annealing (QA). QA (Farhi *et al*., 2000; Das & Chakrabarti, 2008; Smelyanskiy *et al*., 2012) is one of the most accessible quantum algorithms for a computer science audience not versed in quantum computing because of its close ties to classical optimization algorithms such as simulated annealing.

While large-scale universal quantum computers are likely decades away, a variety of special-purpose quantum-computational hardware will emerge in the next few years. Already, early quantum annealers are available, and more sophisticated ones are being designed and built. While certain classes of problems are known to be more efficiently solvable on a universal quantum computer (Nielsen & Chuang, 2001; Rieffel & Polak, 2011), for the vast majority of problems the computational power of quantum computing is unknown. Only now that quantum hardware is becoming available is it possible to empirically evaluate heuristic quantum algorithms such as QA. While there are intuitive reasons why QA may be able to outperform classical methods on some classes of optimization problems, the effectiveness of QA is as yet poorly understood.

In this paper, we introduce the planning community to the key steps involved in compiling planning problems to QA hardware. Our work is the first to explore the use of QA to solve problems arising in

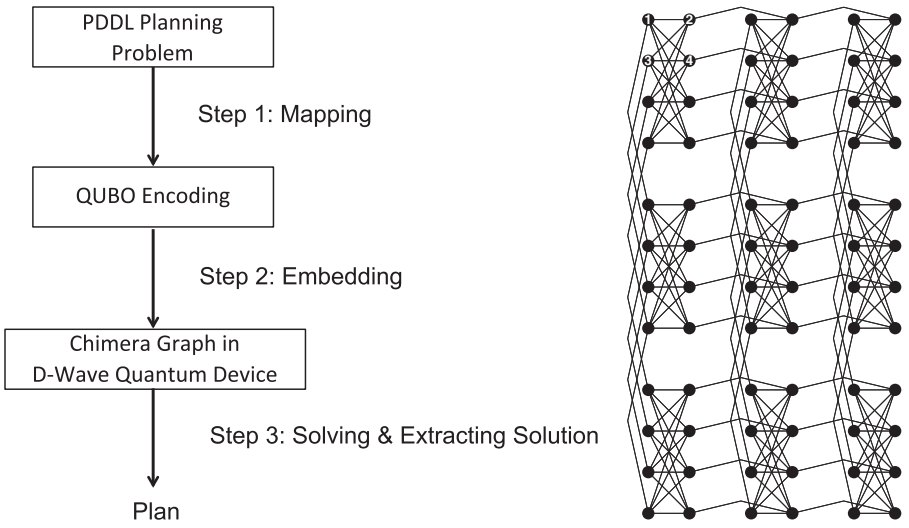

Figure 1 (Left) The main steps in using quantum annealer to solve an application problem. (Right) A schematic diagram from Smelyanskiy *et al.*, (2012) of the (M, L)-Chimera graph underlying D-Wave's architecture. In the (3, 4)-Chimera graph shown, there are $M^2 = 9$ unit cells, each of which is a fully-connected bipartite graph $K_{4,4}$ containing $2L$ qubits

planning and scheduling. This paper is a reworking and deepening of our paper (Rieffel *et al.*, 2015) to target planning and scheduling researchers, rather than quantum computing researchers. Figure 1 shows the main steps in our framework for solving STRIPS planning problems (Fikes & Nilsson, 1972; Ghallab *et al.*, 2004) using a D-Wave Two quantum annealer housed at NASA Ames Research Center: **mapping** the problems to quadratic unconstrained binary optimization (QUBO) and **embedding**, which takes these hardware-independent QUBOs to other QUBOs that matches the specific QA hardware that will be used.

This work examines different mappings of planning problems to QA, giving insight into their relative strengths and weaknesses. We explore properties of these mappings for a parametrized family of planning problems with a strong scheduling component based on graph coloring (Rieffel *et al.*, 2014), and describe results on the D-Wave Two, enabling a comparison of the performance under these mappings. While the immaturity of the technology means that current results are limited, the significant performance differences that result from different compilation approaches suggest that subtle issues are at play in determining the best compilation approaches for quantum annealers.

2 Ingredients of quantum annealing

QA can be applied to any optimization problem that can be expressed in QUBO. QA is motivated by the possibility that quantum effects such as tunneling allow for efficient exploration of the cost-function landscape in ways unavailable to classical methods.

2.1 Input for quantum annealing: quadratic unconstrained binary optimization problems

QUBO problems are minimization problems with cost functions of the form

$$q(z_1, \ldots, z_N) = -\sum_{i=1}^{N} h_i z_i + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} J_{i,j} z_i z_j, \tag{1}$$

where the z_i are 0–1 binary variables. A QUBO can be translated to an Ising Hamiltonian, the actual input a quantum annealer takes, through the change of variables $z_i = \frac{1}{2}(s_i + 1)$.

Before describing the more complex QUBO mappings for general STRIPS planning problems, we give an example of a simple mapping from graph coloring to QUBO.

Example 1. *Mapping of Graph Coloring, in which all vertices are to be colored so that any two vertices connected by an edge have different colors, to QUBO.* Let $G = (V, E)$ be a graph with $n = |V|$ vertices,

where E is the set of edges. The QUBO problem corresponding to the graph coloring problem with k colors on graph G will have kn binary variables, z_{ic}, where $z_{ic} = 1$ means that vertex i is colored with color c, and $zic = 0$ means it is not. The QUBO contains two penalty terms:

$$\sum_{i=1}^{n}\left(1-\sum_{c=1}^{k} z_{ic}\right)^2 + \sum_{\{i,j\}\in E}\sum_{c=1}^{k} z_{ic}z_{jc}.$$

The first corresponds to the constraint that each vertex must be colored by exactly one color: $\sum_{c=1}^{k} z_{ic} = 1$. The second corresponds to the constraint that two vertices connected by an edge cannot be colored with the same color. For each edge $\{i, j\}$, we have a term $\sum_{c=1}^{k} z_{ic}z_{jc}$.

2.2 Embedding quadratic unconstrained binary optimizations in specific quantum annealing hardware

As outlined for planning in Figure 1, once an application instance has been mapped to QUBO, a second step is required to compile it to the specific quantum annealing hardware that will be used. Typically, each quantum device has a set of physical quantum bits (qubits) that are linked in a certain way. Ideally, each binary variable z_i in a QUBO formula would be represented by a single qubit q_i of the machine. Hardware constraints place limits, however, on which qubits can be connected to which other qubits.

The term $J_{i,j}z_iz_j$ involving qubits q_i and q_j in the QUBO formula 1 introduced above is physically realized by a coupling between those qubits. D-Wave processors use the Chimera architecture in which each qubit is connected to at most six other qubits (Figure 1), so any QUBO variable that appears in more than six terms must be represented by multiple physical qubits when implemented in this architecture. The D-Wave Two used in the experiments has a (8, 4)-Chimera graph architecture, but with three broken qubits that are not used. Other limitations, beyond the degree six constraint, exist as well. Therefore, each logical qubit must be mapped to a connected set of physical qubits, which, together with the connecting edges, is called the logical qubit's *vertex-model*. The overall mapping of logical qubits and couplers to physical ones is called a *model*, as discussed below.

Consider the simple QUBO $z_1z_2 + z_1z_3 + z_2z_3$, which can be graphically represented as a triangle, with the three variables as the vertices and each edge a quadratic term of the QUBO. Ideally, we would represent each variable by a single qubit with hardware connections between each pair so that the QUBO's three quadratic terms can be directly realized in the hardware. In the hardware graph, Figure 1, no three qubits are all mutually connected to each other. At best, the three variables can be represented by four qubits. We may take, for example, the vertex model for z_1 to be $\{1, 2\}$, for z_2 to be $\{3\}$, and for z_3 to be $\{4\}$, so that there is a connection corresponding to each of the three terms in the QUBO: the term z_1z_2 can be implemented using the connection between qubits 2 and 3, the term z_1z_3 between 1 and 4, and the term z_2z_3 between 3 and 4. The connection between qubits 1 and 2 is also used to encourage the two qubits to take the same value as they are meant to represent a single variable.

3 Mapping general strips planning problems to quadratic unconstrained binary optimization

In this section, we describe two different mappings from a general class of planning problems to QUBO. Specifically, we consider STRIPS planning problems, classical planning problems that are expressed in terms of binary state variables and actions. The first mapping takes a time-slice approach. The second approach first maps a planning problem to SAT, and then reduces higher order terms to quadratic terms through a series of gadgets. Our mappings allow both positive and negative preconditions. Like other compilation approaches that map STRIPS planning problems to substrates such as SAT or MILP, we also bound the compilation horizon to an initial value that is found heuristically. When the corresponding QUBO is not solvable, we gradually extend the horizon until a solution is found.

3.1 Time-slice mapping

This mapping from general classical planning problems to QUBO form is a variant of the one developed and described in (Smelyanskiy *et al.*, 2012). This approach shares many similarities with existing

compilation approaches (to SAT, CSP, MILP, etc.) derived from the Plan Graph (Blum & Furst, 1997). Specifically, it presets a horizon L and then encodes the interleaving proposition and action layers up to the preset level L.

If the original planning problem has N state variables x_i and M actions y_j, and we are looking for a plan of length L, then we define a time-slice QUBO problem in terms of $N(L+1)+LM$ binary variables. There are two groups of binary variables. The first group consists of $N(L+1)$ binary variables $x_i^{(t)}$ that indicate whether the state variable x_i is 0 or 1 at time step t, for $t \in \{0, \ldots, L\}$. The second group consists of LM binary variables $y_j^{(t)}$ that indicate whether or not the action y_j is carried out between time steps $t-1$ and t. The total cost function is written as a sum:

$$H = H_{\text{initial}} + H_{\text{goal}} + H_{\text{precond}} + H_{\text{effects}} + H_{\text{no-op}} + H_{\text{conflicts}}.$$

The first two terms capture the initial condition and the goal condition. Let $\mathcal{I}^{(+)}$ be the set of state variables that are 1 in the initial condition and $\mathcal{I}^{(-)}$ be the set of state variables that are initially set to 0. Similarly, let $\mathcal{G}^{(+)}$ (resp. $\mathcal{G}^{(-)}$) be the set of goal variables with value 1 (resp. 0). To capture the requirement that a plan start in the appropriate initial state and meets the goals, we include in the cost function the term

$$H_{\text{initial}} = \sum_{i \in \mathcal{I}^{(+)}} \left(1 - x_i^{(0)}\right) + \sum_{i \in \mathcal{I}^{(-)}} x_i^{(0)}, \quad H_{\text{goal}} = \sum_{i \in \mathcal{G}^{(+)}} \left(1 - x_i^{(L)}\right) + \sum_{i \in \mathcal{G}^{(-)}} x_i^{(L)}.$$

We next add terms to the cost function that penalize a plan if an action is placed at time t, but the prior state does not have the appropriate preconditions:

$$H_{\text{precond}} = \sum_{t=1}^{L} \sum_{j=1}^{M} \left(\sum_{i \in \mathcal{C}_j^{(+)}} \left(1 - x_i^{(t-1)}\right) y_j^{(t)} + \sum_{i \in \mathcal{C}_j^{(-)}} x_i^{(t-1)} y_j^{(t)} \right),$$

where $\mathcal{C}_j^{(+)}$ is the set of positive preconditions for action j and $\mathcal{C}_j^{(-)}$ is the set of negative preconditions. Next, we must penalize variable changes that are not the result of an action. We start with this term, the $H_{\text{no-op}}$ term, that penalizes variable changes:

$$H_{\text{no-op}} = \sum_{t=1}^{L} \sum_{i=1}^{N} \left[x_i^{(t-1)} + x_i^{(t)} - 2 x_i^{(t-1)} x_i^{(t)} \right].$$

This term gives a cost penalty of 1 every time a variable is flipped. Of course, when the effect of an action does result in a variable flipping, we do not want this penalty, so we will make up for this penalty when we add the term that corresponds to the effects of an action. Specifically, we need to penalize if the subsequent state does not reflect the effects of a given action. Let $\epsilon_j^{(+)}$ be the set of positive effects for action j and $\epsilon_j^{(-)}$ the set of negative effects. The penalty if the appropriate effects do not follow the actions is captured by the following term:

$$H_{\text{effects}} = \sum_{t=1}^{L} \sum_{j=1}^{M} \left(\sum_{i \in \epsilon_j^{(+)}} y_j^{(t)} \left(1 + x_i^{(t-1)} - 2 x_i^{(t)}\right) + \sum_{i \in \epsilon_j^{(-)}} y_j^{(t)} \left(2 x_i^{(t)} - x_i^{(t-1)}\right) \right).$$

In order to understand this term, we must consider it together with the no-op term. When $y_j^{(t)} = 1$, the corresponding term for $i \in \epsilon_j^{(+)}$ (resp. $i \in \epsilon_j^{(-)}$), taken together with the no-op term, can be written $\left(1 + 2x_i^{(t-1)}\right)\left(1 - x_i^{(t)}\right)$ (resp. $\left(3 - 2x_i^{(t-1)}\right) x_i^{(t)}$ for negative effects), resulting in a positive penalty unless $x_i^{(t)} = 1$ (resp. $x_i^{(t)} = 0$). By using this form we have corrected for the corresponding no-op term.

3.1.1 Parallel plans

Classical planners often allow for parallel plans in which more than one action can take place at one time if those actions could have been done in any order. Encodings that allow parallel plans are often significantly smaller due to the big reduction in the preset horizon value L. The QUBO encoding described so far works fine for domain with linear plans, but when more than one action can take place at a given time, we are in

danger of over-correcting for the no-op term. If multiple actions at the same time have the same effect, the H_{effects} term will add a term for each of those actions, thus overcompensating for the no-op penalty. To avoid overcompensating, we penalize multiple actions at the same time having the same effect, discouraging all such actions to ensure that two actions that conflict in the sense that positive preconditions of one overlap with negative effects of the other or vice versa, and to avoid overcompensating, we include the penalty:

$$H_{\text{conflict}} = \sum_{t=1}^{L}\sum_{i=1}^{N}\left(\sum_{\left\{j \mid i\in C_j^{(+)}\cup \epsilon_j^{(-)}\right\}}\ \sum_{\left\{j'\neq j \mid i\in \epsilon_{j'}^{(-)}\right\}} y_j^{(t)}y_{j'}^{(t)} + \sum_{\left\{j \mid i\in C_j^{(-)}\cup \epsilon_j^{(+)}\right\}}\ \sum_{\left\{j'\neq j \mid i\in \epsilon_{j'}^{(+)}\right\}} y_j^{(t)}y_{j'}^{(t)}\right).$$

3.1.2 Encoding size improvements

Although for explanatory purposes it was useful to include variables for the state at time $t = 0$, those specified by initial conditions can be set ahead of time, so that we don't need to include the H_{initial} term. The same is true of the H_{goal} term. We can also replace all of their occurrences in $H_{\text{no–op}}$, H_{precond}, and H_{effect} with these set values to simplify those constraints. Furthermore, reachability and relevant analysis starting from the initial and goal states, preprocessing techniques employed by compilation-based planners such as Blackbox (Kautz & Selman, 1999) and GP-CSP (Do & Kambhampati, 2001), can be used to remove or preset the values of variables in different layers. These simplifications result in modified terms $H'_{\text{no–op}}$, H'_{precond}, and H'_{effects}. In addition, as in our setting we have followed the convention that preconditions must be positive, we can use a simpler version of the H_{precond} term: $H'_{\text{precond}} = \sum_{t=1}^{L}\sum_{j=1}^{M}\sum_{i\in C_j^{(+)}}\left(1-x_i^{(t-1)}\right)y_j^{(t)}$. For the scheduling problems we consider here, the QUBO simplifies to $H = H'_{\text{no–op}} + H'_{\text{precond}} + H_{\text{effacts}} + H_{\text{conflict}}$.

3.2 Conjunctive normal form-based mapping

We also explored first mapping planning problems to SAT (in conjunctive normal form (CNF) form) and then using a standard approach to map the resulting CNF expression to QUBO. A SAT's CNF expression over n Boolean variables $\{x_i\}$ consists of a set of clauses $\{C_a\}$, each with k variables, possibly negated, connected by logical ORs: $b_1 \vee b_2 \vee , \ldots, \vee b_k$, where $b_i \in \{x_1, x_2, \ldots, x_n, \neg x_1, \neg x_2, \ldots, \neg x_n\}$. The number of variables k in the clause can vary from clause to clause. The clauses are connected by an AND operator, meaning all of the clauses must be satisfied.

We used the first of four PDDL-to-CNF translators built into the SATPLAN planner (Kautz, 2004). This "action-based" encoding starts with a time-slice encoding approach and then further removes all variables representing state variables while adding constraints that capture the relationships between actions in consecutive time steps that were previously enforced by relationships between actions and state variables. We chose this encoding because it tends to produce the smallest SAT encodings, which are more likely to translate to QUBOs that embed in the current D-Wave machine hardware.

To convert a CNF instance to QUBO, we first transform it to a polynomial unconstrained binary optimization (PUBO), a generalization of QUBO, which allows terms with higher than quadratic degree. Each clause of a CNF instance translates to a PUBO instance that is the conjunction of the negation of all the literals in that clause; a negative literal is replaced by a binary variable and a positive literal is replaced by the difference of 1 and the corresponding binary variable. For example, the CNF clause $(x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4)$ would correspond to the PUBO term $(1 - x_1)x_2x_3(1 - x_4)$.

We reduce higher degree terms in a PUBO to quadratic using an iterative greedy algorithm related to one described in (Boros & Hammer, 2002). At each step, a pair of variables that appears in most terms is replaced by an ancilla variable corresponding to their conjunction. We add a penalty term to ensure the ancilla variable corresponds to this conjunction. For example, for the term $x_1x_2x_3$, we may introduce an ancilla variable y_{12} to replace x_1x_2, and add a penalty term $3y_{12} + x_1x_2 - 2x_1y_{12} - 2x_2y_{12}$, which is 0 if $y_{12} = x_1x_2$ and >0 otherwise. We remove the term $x_1x_2x_3$ from the PUBO, replacing it with

$y_{12}x_3 + 3y_{12} + x_1x_2 - 2x_1y_{12} - 2x_2y_{12}$. We use a penalty weight greater by 1 than the sums of the magnitudes of the positive and negative coefficients of the terms the ancilla is used to reduce (Babbush *et al.*, 2013) to ensure that the constraint-satisfying solutions have lower total cost than the constraint-violating solutions. This procedure is repeated until the resulting PUBO is quadratic.

4 Experimental setup

Many planning applications include scheduling aspects (Chien *et al.*, 2012). Scheduling, which deals with assigning resources and time to tasks while taking into account constraints, is in itself an important problem. Certain classes of scheduling problems correspond to graph coloring. For example, a scheduling problem with a set of tasks and constraints such that any pair of tasks competing for the same resource cannot be assigned the same time-slot can be phrased as vertex coloring, a well-known NP-complete problem. Specifically, the chromatic number (the smallest number of colors needed) represents the smallest number of time-slots needed to complete a corresponding schedule, thus representing the minimum makespan.

To evaluate our approach, we use a benchmark set of planning instances based on graph coloring (Rieffel *et al.*, 2014) that consist of parametrized families of hard planning problems. Having parametrized families enables the investigation of scaling behavior using small problems, which is crucial for evaluating early technology that is not mature enough to run real-world problems. Because of the overhead in mapping and embedding planning problems, even the smallest IPC problems (International Planning Competition, 2004) are more than an order of magnitude too large to be run on the current D-Wave device.

4.1 Vertex coloring as planning

A scheduling problem in which no pair of tasks can be assigned the same time-slot is analogous to a coloring instance in which the tasks are vertices, conflicts are edges, and the minimum makespan is the chromatic number, which is the minimum number of colors necessary to color each of the vertices such that no two adjacent ones have the same color. Given an undirected graph $G = \{V, E\}$ with n vertices and k colors, the vertex coloring problem asks for a solution in which (1) all vertices are colored and (2) any pair of vertices connected by an edge is colored differently. The corresponding planning problem is as follows: for each vertex v there are

- k actions a_v^c representing coloring v with color c,
- A "goal variable" s_v^g representing whether or not v has been colored at all, and
- A "state variable" s_v^c representing whether or not v has been colored with the color c.

Let $C(v)$ be the set of neighboring vertices that are connected to v by an edge. For each action a_v^c, there are $|C(v)| + 1$ preconditions: $(1) s_v^g = F$, indicating that v is not already colored; and (2) for each $w \in C(v)$, $s_w^c = F$, guaranteeing that none of neighboring vertices are already colored with color c. Each action a_v^c has two effects: $s_v^g = T$ and $s_v^c = T$. In the initial state, none of the vertices are colored: $\forall v, \forall c \in [k]: s_v^g = F$, and $s_v^c = F$. The goal state requires that all vertices are colored: $\forall v: s_v^g = T$. A plan is a sequence of n actions, each of which colors a vertex v.

4.1.1 Problem generation

We parametrically generate instances using the Erdös-Rényi model of random graphs $G_{n,p}$, where n is the number of vertices and p is the probability of an edge between each pair of vertices, using an extension of Culberson et al.'s (1995) graph generator program. Our extension generates PDDL files (Rieffel *et al.*, 2015), at the phase transition $c = pn = 4.5$ (Achlioptas & Friedgut, 1999; Dubois & Mandler, 2002; Achlioptas & Moore, 2003; Coja-Oghlan, 2013). We preset the number of colors to $k = 3$ and for that k value the maximum problem size that we can embed in the D-Wave Two machine with 509 qubits is $n = 16$. For each of $n = 8, 9, \ldots, 16$, we use 100 solvable problem instances at the phase transition for each size. For each generated instance, we then generate three different QUBOs, each described in the previous sections: (i) direct mapping; (ii) time-slice mapping; and (iii) CNF-based mapping.

4.1.2 Embedding

From a mapped QUBO instance, we generate a vertex model by running D-Wave's heuristic embedding software (Cai *et al.*, 2014) on the mapped QUBO instance, using the software's default parameters. The output of the embedding software is a set of pairwise-disjoint, connected vertex models $\{C_i\}$ in the hardware graph corresponding to the variables $\{z_i\}$ in the original QUBO, which will then be converted to Ising form to be run on the D-Wave machine. Before running, the Ising is rescaled so that all coefficients are between $[-1, 1]$. We performed our own parameter setting following (Rieffel *et al.*, 2015), rather than using D-Wave's defaults.

4.1.3 Solving

All quantum annealing runs were on the 509-qubit D-Wave Two machine housed at NASA Ames with an anneal time of 20 microseconds. For each embedded QUBO instance, we performed 450,000 anneals for 10 gauges (i.e. local symmetry transformations that leave the objective function invariant but reduce the effect of hardware biases(Perdomo-Ortiz *et al.*, 2015)), for a total of 450 000 anneals per instance.

We consider only solvable instances, so that the minimum value of all our QUBO instances is zero. For each embedded instance, once we obtain the 450,000 results from the run, we check how many times the minimal value was obtained, which gives us the probability of solution r. We then compute the expected number of runs $R=\frac{\ln(1-0.99)}{\ln(1-r)}$ required to obtain a 99% success probability, multiplied by the anneal time of 20 microseconds, and report $20\times R$ microseconds, the expected total anneal time to 99% success probability. We are effectively using a 0.9 seconds cutoff time as the expected anneal time when only one anneal solves is 0.9 seconds. Given that classical planners solve these problems in <0.1 seconds, with the best planners for these problems solving them in <0.01 seconds (Rieffel *et al.*, 2014), this cutoff time seems reasonable.

We report the median expected total anneal time across 100 instances, with error bars corresponding to the 35th and 65th percentiles. Each data point shown therefore represents 45 million anneals. Although the total annealing time for each point is only 90 seconds, because the process to read-out the state of all qubits (i.e. solution extraction) takes considerably longer than the anneal time, and because of shared use of the machine, the wall clock time to obtain a single data point is hours rather than minutes. Finding the embedding, by far the longest step in the process, can take minutes for the largest instances, but fortunately needs to be performed only once per QUBO instance.

5 Results and analysis

Figure 2 shows the relative performance, in terms of median expected total annealing time for 99% success, of the D-Wave Two on the family of graph coloring-based planning problems described above. When at least half of the instances do not solve within the 0.9 seconds effective cutoff time, we no longer show the point. For the CNF mapping, that happens by problem size 11, and for the time-slice instances by problem size 13. The figure also shows the performance using a direct map of graph coloring to QUBO. As expected, this direct mapping performs better than both general mappings for planning problems as it is more compact.

There is a substantial difference between the performance on the time-slice instances and the CNF instances, with the median expected total annealing time to achieve 99% success being about a factor of 5 greater for the CNF instances than for the time-slice instances (Figure 2). The scaling for the time-slice approach is also significantly better than for the CNF approach, with an α-value of 1.37 rather than 1.76 (though the scaling is estimated on very few data points). Rieffel *et al.*, (2015) compared several properties of both the mapped and embedded QUBOs for the two mappings, but these simple properties were all sufficiently similar across the two mappings that they could not account for so marked a difference in performance.

We took a deeper look at the distributions of various simple properties related to the mapped and embedding QUBOs arising from the two mappings. We also show the distributions for the direct map for comparison. In the mapped QUBOs, we looked at the distribution of of vertex degrees (the number of

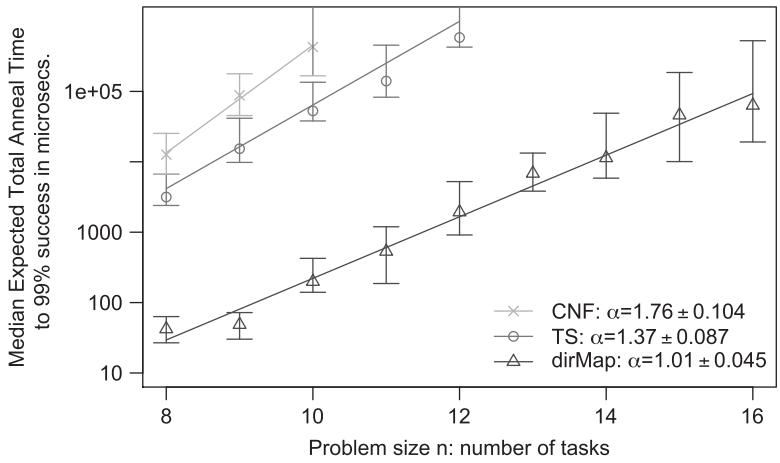

Figure 2 Comparison of the median expected total anneal time to 99% success for the two mappings. Each data point shows the median expected total annealing time to achieve 99% success over the 100 problems of each size given on the x-axis. The error bars are at the 35th and 65th percentiles. When at least half of the instances do not solve within the 0.9 seconds effective cutoff time, we no longer show the point. Also, when fewer than 65% solve, the top of the error bar is indeterminate, as happened for the last point shown in both the conjunctive normal form (CNF) and time-slice (TS) series

quadratic terms in which a variable z_i appears in the mapped QUBO). As can be seen in Figure 3a, the histograms for the time-slice and CNF mappings are very similar, whereas they differ markedly from the histograms for the direct map. In the embedded QUBOs, we looked at the distribution of the vertex model sizes (Figure 3b), and also the distribution of the graph diameter of the vertex models (not shown), but found little difference between the distributions for the time-slice and CNF embedded QUBOs. Therefore these properties can contribute at most a small amount to the performance difference between the two mappings.

We begin to see differences when we look at distributions of the coefficients in the mapped QUBOs. Figure 3c shows histograms of the coefficient of the mapped QUBOs (Equation (1)), converted to Ising, and rescaled so that all of the h_i and J_{ij} coefficients are between $[-1, 1]$. Let $j_i = \sum_j J_{ij}$. Figure 3d shows a histogram of the h_i and j_i. Both histograms show significant differences between the two mappings.

Another potential origin of the performance difference is the topology of the vertex models. A quick analysis showed that for all three mappings nearly all (>99%) of the vertex models of the embedded QUBOs are trees. A further classification of the graph structures, and their relative frequencies, could potentially lead to greater insight.

6 Conclusions and future work

We show how quantum annealing can solve planning problems via mapping to QUBO. We introduced two general mapping techniques and applied them to planning problems based on graph coloring. We ran these problems on an early quantum annealer and saw significant performance differences. We began an investigation of various properties of the mapped and embedded QUBOs to understand which properties do and do not contribute to the performance differences.

One of the biggest open questions in quantum computing is the breadth of its applications. Although certain quantum algorithms have been proven to outperform classical algorithms on classes of problems of practical interest, the potential advantage of quantum methods for the vast majority of problems remains unexplored. Many of the most useful classical algorithms in use today are heuristic algorithms, which have not been mathematically proven to outperform other approaches, but have been shown to be more effective empirically. Only recently, with the advent of early quantum annealers, could empirical investigation of quantum techniques begin. Major hardware development efforts are underway to build better quantum computational hardware. In order to fully explore the potential of this hardware, we must understand how

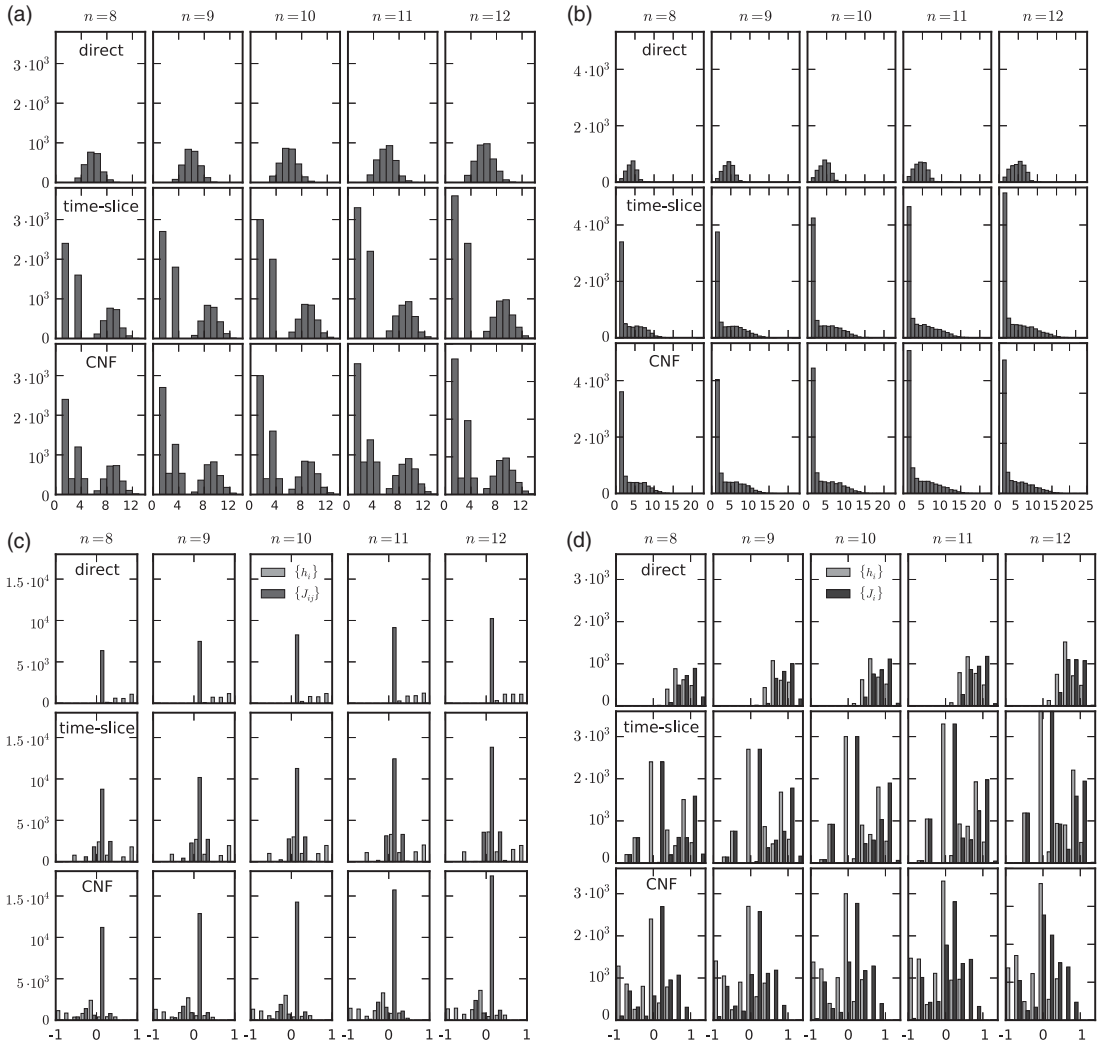

Figure 3 Analysis and comparison of mapped and embedded Quadratic Unconstrained Binary Optimization (QUBO) features for each problem size under three mappings: direct mapping, time-slice mapping, and conjunctive normal form (CNF) mapping (a) Vertex degree histogram. Histograms of the vertex degrees for the mapped QUBO graphs. (b) Vertex model size histogram. Histograms of the vertex model sizes in the embedded QUBOs. (c) QUBO coefficient histogram. Histograms of the h_i and J_{ij} in the mapped QUBOs. (d) Histogram of h_i and $j_i = \Sigma_j J_{ij}$. Histograms of the h_i and $j_i = \Sigma_j J_{ij}$ in the mapped QUBOs.

best to compile practical problems to a form that is suitable for quantum hardware. Early quantum annealing hardware can handle only small instances. At this early stage there is no advantage in solving STRIPS planning problems via quantum annealing over classical compilation approaches such as SAT, CSP, or MILP. Nevertheless, by analyzing the results obtained under these limitations, we can gain insights into the best programming and compilation techniques for quantum annealers, and ultimately into the potential of quantum annealing to solve problems of practical interest in planning and scheduling and beyond.

Acknowledgments

The authors are grateful to Zhihui Wang for helpful discussions. This work was supported in part by the Office of the Director of National Intelligence (ODNI), the Intelligence Advanced Research Projects Activity (IARPA), via IAA 145483; by the AFRL Information Directorate under grant F4HBKC4162G001. The views and conclusions contained herein are those of the authors and should not

be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ODNI, IARPA, AFRL, or the US Government. The US Government is authorized to reproduce and distribute reprints for Governmental purpose notwithstanding any copyright annotation thereon. The authors also like to acknowledge support from the NASA Advanced Exploration Systems program and NASA Ames Research Center and NASA grant NNX12AK33A.

References

Achlioptas, D. & Friedgut, E. 1999. A sharp threshold for k-colorability. *Random Structures and Algorithms* **14**(1), 63–70.

Achlioptas, D. & Moore, C. 2003. Almost all graphs with average degree 4 are 3-colorable. *Journal of Computer and System Sciences* **67**(2), 441–471.

Babbush, R., O'Gorman, B. & Aspuru-Guzik, A. 2013. Resource efficient gadgets for compiling adiabatic quantum optimization problems. *Annalen der Physik* **525**(10–11), 877–888.

Blum, A. & Furst, M. 1997. Planning through planning graph analysis. *Artificial Intelligence Journal* **90**, 281–330.

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

Cai, J., Macready, B. & Roy, A. 2014. A practical heuristic for finding graph minors. arXiv:1406:2741.

Chien, S., Johnston, M., Frank, J., Giuliano, M., Kavelaars, A., Lenzen, C., Policella, N. & Verfailie, G. 2012. A generalized timeline representation, services, and interface for automating space mission operations. In *12th International Conference on Space Operations*.

Coja-Oghlan, A. 2013. Upper-bounding the k-colorability threshold by counting covers. arXiv:1305.0177.

Culberson, J., Beacham, A. & Papp, D. 1995. Hiding our colors. In *Proceedings of the CP95 Workshop on Studying and Solving Really Hard Problems*. 31–42.

Das, A. & Chakrabarti, B. K. 2008. Colloquium: quantum annealing and analog quantum computation. *Reviews of Modern Physics* **80**, 1061–1081.

Do, M. B. & Kambhampati, S. 2001. Planning as constraint satisfaction: solving the planning graph by compiling it into CSP. *Artificial Intelligence Journal* **132**(2), 151–182.

Dubois, O. & Mandler, J. 2002. On the non-3-colourability of random graphs. arXiv:math/0209087.

Farhi, E., Goldstone, J., Gutmann, S. & Sipser, M. 2000. Quantum computation by adiabatic evolution. arXiv:quant-ph/0001106.

Fikes, R. E. & Nilsson, N. J. 1972. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* **2**(3), 189–208.

Ghallab, M., Nau, D. & Traverso, P. 2004. *Automated Planning: Theory & Practice*. Elsevier.

International Planning Competition 2004. *The International Planning Competition Website*. `http://icaps-conference.org/index.php/Main/Competitions`.

Kautz, H. 2004. SATPLAN04: planning as satisfiability. Working Notes on the Fourth International Planning Competition (IPC-2004), 44–45.

Kautz, H. A. & Selman, B. 1999. Unifying SAT-based and graph-based planning. In *Proceedings of IJCAI'1999*.

Nielsen, M. & Chuang, I. L. 2001. *Quantum Computing and Quantum Information*, Cambridge, Cambridge University Press.

Perdomo-Ortiz, A., Fluegemann, J., Biswas, R. & Smelyanskiy, V. N. 2015. A performance estimator for quantum annealers: gauge selection and parameter setting. *arXiv preprint arXiv:1503.01083*.

Rieffel, E. G. & Polak, W. 2011. *A Gentle Introduction to Quantum Computing*, Cambridge, MA, MIT Press.

Rieffel, E. G., Venturelli, D., Hen, I., Do, M. & Frank, J. 2014. Parametrized families of hard planning problems from phase transitions. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI-14). 2337–2343*.

Rieffel, E. G., Venturelli, D., O'Gorman, B., Do, M. B., Prystay, E. M. & Smelyanskiy, V. N. 2015. A case study in programming a quantum annealer for hard operational planning problems. *Quantum Information Processing* **14**(1), 1–36.

Smelyanskiy, V. N, Rieffel, E. G, Knysh, S. I, Williams, C. P, Johnson, M. W., Thom, M. C., Macready, W. G. & Pudenz, K. L. 2012. A near-term quantum computing approach for hard computational problems in space exploration. arXiv:1204.2821.