Introduction to Quantum Optimization using D-WAVE 2X

Thamme Gowda

January 27, 2018

Development of efficient algorithms has been a goal ever since computers were started to use for solving real world problems. The complexity of algorithms are classified into various classes of growth, such as linear, quadratic, exponential etc. These classes help in estimating the time or memory required to run the algorithms in terms of input size. Although a tremendous progress is made in developing efficient solutions to problems in general, there are still some hard problems that require exponentially more resources as the input size grows. The 'traveling salesman problem' is a well known example to this case. To such problems, any polynomial complexity growth solution would be considered as an efficient solution, however so far in literature and practice there is no report of any such algorithms. Some researchers question if efficient solutions to such problems are possible.

Since the early days of computers, the digital computers have been the reference for assessing the computational complexity. These will be referred as classical computers in the rest of the writing. Classical computers work by encoding data and instructions to binary states: 0 and 1. Classical computers such as laptops, servers, and smartphones are practically Turing complete - they can simulate a turing machine. These machines provide rich set of machine instructions and programming APIs to solve most of the real world problems. However, for some problems such as combinatorial optimization problems they require exponentially more time as the input size increases. Since there are no algorithms whose complexity in polynomial complexity class, such problems are believed to be in NP complexity class. The questions to be asked now are 1) Is this the dead end for NP problems on classical computers? 2) What other kinds of computing is possible? and 3) Can other kinds of computing help us solve these hard problems quickly? I do not know if we will ever invent a efficient solutions to those hard problems using classical computers. The answer to the second question, Quantum computing, is the other kind of computing interested to us in this tutorial. These computers function based on quantum-mechanical phenomena such as superposition and entanglement of quantum bits. Question 3 will be the topic for the remainder of this report.

In classical computers we use binary digits, or in short 'bits', where each bit is clearly either 0 or 1 at any time. In contrast quantum bits, or 'qbits' in short, may be entangled state of 0 and 1 at times. An equivalent turing complete quantum computer is purely theoretical at the time of writing, and I am not interested in porting perfectly good solutions from classical computers to quantum computers. There are computers believed to be practical quantum computers (maybe not turing complete) such as D-Wave 2X, which offers instructions for solving combinatorial optimizations problems. Since the computational complexity of best algorithms that solve combinatorial optimization problems is believed to be in NP class of classical programming model, exploring efficient solutions using quantum programming model is particularly interesting.

I have picked two combinatorial problems - partition problem and binary integer linear programming (BILP) problem - for providing step-by-step guide, which are covered in 3.1 and 3.2 respectively. However, before jumping to it, we need to understand two optimization models, namely, Quadratic Unconstrained Binary Optimization (QUBO) and Ising Model(IM), covered in section 1. We will see that IM problems can be directly encoded to quantum hardware, and show it by actually implementing number partition problem. We will also see that QUBO is convenient to directly express some other kinds of problems, for example Binary Integer Programming (BILP), and learn how to convert QUBO to IM. Section 2 offers some insights into DWAVE hardware and software since it is necessary for running our example problems. Since we are still in the very early stages of quantum programming and thus it comes with a few limitations. Section 4 share some insights of limitations, and help you set expectations right.

1 Optimization Models

Things you should either agree or debate with me to make the rest of the tutorial consistent.

- We are trying to solve Combinatorial Optimization problems.
- Optimization Problem = find the extreme value such as global minima or global maxima of a function.
- Function = a mathematical structure that takes argument and maps it to a value.
- Combinatorial Optimization Problem = find which combination of arguments that results in global extrema of a function. Sometimes the arguments as also called as variables.
- Discrete Optimization = Arguments or variables can be discrete items such as integers or items in a finite set such as boolean values. Let us understand boolean variables first.
- These optimization problems are hard to solve. So far, we don't have efficient and precise solutions that run on classical computers.
- We are excited to give quantum programming model a spin!

The question you might think right now, is - what makes quantum programming work faster than the classical programming? The answer is - (1) Ising Spin Model and (2) Quantum Annealing.

Ising Model helps us to express (some of) our optimization problems directly, especially those that have boolean decision variables. This model is pretty old, it was developed to study magnetism about a century ago. Now we have found a clever use of it in quantum programming.

An Ising model problem consists of binary variables, $s_i \in \{-1, +1\}$, with the cost or energy function:

$$E(s_1, s_2, ...s_n) = \sum_i h_i s_i + \sum_{<i,j>} J_{<i,j>} s_i s_j \tag{1}$$

So, the first step for us is to take our optimization problem and express it in Ising Model such that the variables combinations yielding global optima of our original problem will produce global minima of the above energy function. This mapping is tricky, and the core/heart of quantum optimization.

Now, let us suppose we take a problem and express it as ising problem. Then, how can we efficiently find out the arguments that yields minima of this energy function? Answer is quantum annealing. This should sound similar to simulated annealing! If we have an Ising model problem, we can use D-Wave's API to encode(or embed) our problem into their hardware (a mesh of qbits with couplers) and then let their quantum annealer magic find the global optima in constant annealing time. (I will drop the word magic when I actually understand how quantum physics work, for now, its not my field of science.)

Looking closely at equation 1, we can say

- $s_1, s_2, ...s_n$ are variables in ising model. These will be quantum bits when we embed the problem to quantum hardware. Note that these are the output variables, we never programmatically set the values of these, instead the annealer will set it during the annealing process and we will read the final decisions. The values will be either +1 or -1 when we read it (after annealing).
- h_i variables are the inputs named strengths. They exists one per qbit. We are going to set the values depending on the problem we are solving (more on this in the examples later).
- $J_{<i,j>}$ variables are inputs named couplings, we are going to set values depending on the problem. $< i, j >$ means that these two quantum bits are neighbors and there exists a coupler between them, and $i \neq j$ (no coupler between itself). Not all pairs of qbits maybe coupled in the quantum hardware, which is true in our case of D-Wave hardware.

Quadratic Unconstrained Binary Optimization (QUBO)
Physicists loved Ising Model (for explaining magnetic fields, and now the quantum mechanics), but pure and applied mathematicians studied a slightly different model called QUBO. A QUBO problem consists of binary variables $x_i \in \{0, 1\}$ with the cost or energy function:

$$E(x_1, x_2, ...x_n) = \sum_{i,j} x_i Q_{ij} x_j \tag{2}$$

QUBO to Ising Conversion:
QUBO and Ising problems can be transformed from one to other using a simple transformation $s_i = 2x_i - 1$ where $x_i \in \{0, 1\}$ and $s_i \in \{-1, +1\}$

2 D-wave 2X Quantum Computer

2.1 Chimera Graph

A fully connected graph has an edge between any two pairs of nodes in graph. If a graph has fewer edges than the fully connected graph, then it is called sparse graph. There is only one fully connected graph topology for a given set of nodes, but many sparse graph topology. A particular kind of sparsely connected graph in which nodes are placed on a 2D plane and are connect to neighbors only in a fashion as shown in Figure 1 is called chimera graph.

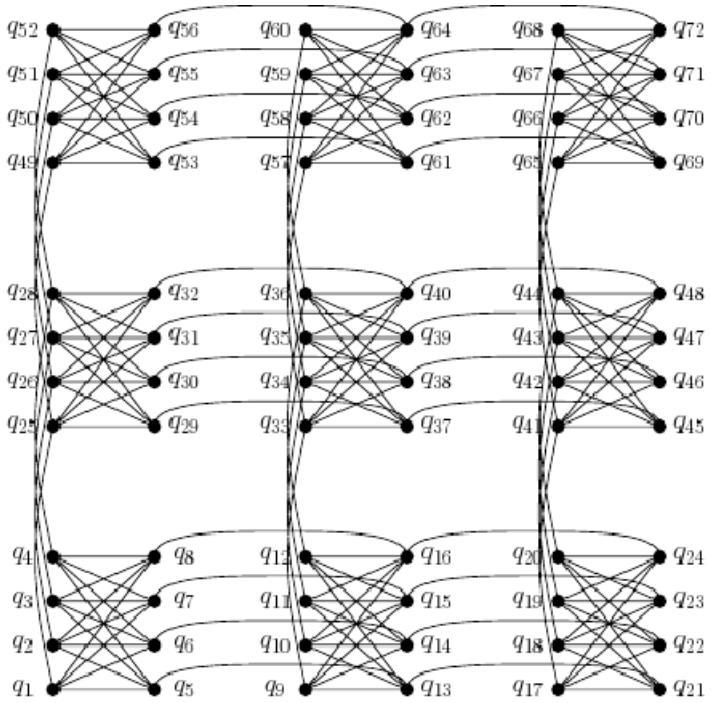

Figure 1: Chimera Graph of 72 nodes. Courtesy: https://arxiv.org/pdf/1407.2887

2.2 Solvers

D-Wave offers two variations of solvers for optimization problems, namely a local simulated solver and the remote hardware solver. A local simulated solver simulates quantum annealing for up to 128 qbits. The hardware solver claims to do the actual quantum annealing and supports larger number of qbits (our lab had a 1098 qbit machine).

2.3 APIs

D-Wave offers programming APIs in python, C++ and Matlab languages. I have used python api which was based on python version 2.7. Refer to the D-Wave documentation available at your site to learn about the setup instructions.

3 Examples

3.1 Partition Problem

Partition problem is considered to be a simple NP-Complete problem. Coincidentally, it is one of the problem that can be easily formulated as ising model problem. In partition problem, we are given with a set of numbers, the goal is to partition the set into two sets such that the sum of two sets are equal.

Example: $S = \{1, 2, 3, 4\}$ Partitions: $A = \{1, 4\}$ and $B = \{2, 3\}$. Note that both the partitions have a total value of 5. There could be more than one way of partitioning the set if the input permits!

More formally, the input set $S = \{x_1, x_2, x_3, ...x_n\}$. The task is to split S into A and B, such that $Sum(A) = Sum(B)$

Let us express this partition problem in ising model. First, for each item, x_i, in the set, let us introduce s_i a decision variable which captures the decision whether x_i should go to the partition A or B. When we have an ideal split, the sum of items in the partition A should be same as sum of B, so that should be the ground state or lowest energy state in the ising model energy function, lets us call it as E_{optima}. There shall be multiple E_{optima}s for this problem.

For any combination of decisions $s_1, s_2, ..s_n$ that cause incorrect partition, the sum of partitions will not be same, then the overall energy of ising function should be higher than the E_{optima}.

In other words, $(Sum(A) - Sum(B))^2 = 0$ for correct partitions, and $(Sum(A) - Sum(B))^2 \geq 0$ for incorrect partitions.

$$E = (\sum_{i \in A} x_i - \sum_{j \in B} x_j))^2$$

Simplify this by using decision variables $s_i = +1$ if $x_i \in A$ and $s_i = -1$ if $x_i \in B$ The energy of our problem is $E = (\sum_{i=1}^{N} s_i x_i)^2$. This is one step closer to the equation 1, but we need some more reduction.

$$E = (\sum_{i=1}^{N} s_i x_i)^2 = \sum_{i=1}^{N} s_i^2 x_i^2 + \sum_{i,j;i \neq j}^{N} s_i x_i x_j s_j$$

Now, there is an interesting reduction here. We dont really need absolute value of E, our only interest is that the Minimum value of E should be our global minima. Since $s_i \in \{-1, +1\}$, $s_i^2 = 1$, thus $\sum_{i=1}^{N} s_i^2 x_i^2 = \sum_{i=1}^{N} x_i^2$, and it is constant value for all the combinations of $s_1, s_2, ...s_N$, so we can drop that term.

The final reduction, therefore, is:

$$E \approx \sum_{i,j;i \neq j}^{N} s_i x_i x_j s_j$$

This is in the same form as equation 1 (just assume $h_i = 0, \forall i$), which means we are now ready to use a quantum annealer.

But, there is one more thing, in this equation there are interactions between every possible pair of s_i and s_j, but in reality, the quantum bit couplers have a chimera topology. So, to overcome this we will use an extra step of connecting multiple physical qbits as one logical qbit. This can be done using a library functions named `find_embedding()`, `embed_problem()`, and `unembed_answer()`.

Now that we did all the math needed, let us test it out with D-Wave's Python API.

```
from dwave_sapi2.local import local_connection
from dwave_sapi2.remote import RemoteConnection
from dwave_sapi2.core import solve_ising
from dwave_sapi2.util import get_hardware_adjacency
from dwave_sapi2.embedding import find_embedding, embed_problem, unembed_answer

# Local solver is a simulated quantum annealer
print 'Available Local Solvers:', local_connection.solver_names()
loc_solver = local_connection.get_solver("c4-sw_optimize")

# Remote solver is the real quantum annealer, get URL and tokens of yours
# url = "https://<...>.isi.edu/sapi"
# token = ""
# rem_solver = RemoteConnection(url, token).get_solver("DW2X")

def solve_partitioning(S, solver, num_reads=10):
  """Uses Quantum Annealer to Solve Partitioning Problem """
  h = [0] * len(S)  # h are all zeros
  J = {} # couplers
  for i in range(len(S)):
    for j in range(i+1, len(S)):
      J[i,j] = S[i] * S[j]

  # Embed this problem to DWave's Chimera.
  A = get_hardware_adjacency(solver)  # currently active bits and couplers
  # Search for embeddings of J which uses only the active couplers
  embeddings = find_embedding(J, A, verbose=0)

  # Now embed the problem with whatever we found in previous step
  [h0, j0, jc, new_embeds] = embed_problem(h, J, embeddings, A)
  # j0 has updated coupler weights.
  # jc has cloning of extra qbits to improve connectivity. We need both
  emb_j = j0.copy()
  emb_j.update(jc)
  result = solve_ising(solver, h0, emb_j, num_reads=num_reads)
  sols = unembed_answer(result['solutions'], new_embeds, 'minimize_energy', h, J)
  res = []   # post process the result
  for i, sol in enumerate(sols):
    energy = result['energies'][i]
    prob = 1.0 * result['num_occurrences'][i] / num_reads
    sol = [bit for bit in sol if bit != 3]  # unused  bits have value of 3
    res.append({'E': energy, 'P': prob, 'S': sol})
  return res

def print_solutions(sols, S):
  for i, sol in enumerate(sols):
    assert len(sol['S']) == len(S) # decision variables and items in list match
    A = [x_i for s_i, x_i in zip(sol['S'], S) if s_i == 1]
    B = [x_i for s_i, x_i in zip(sol['S'], S) if s_i == -1]
    A_sum, B_sum  = sum(A), sum(B)
    OPT = max(A_sum, B_sum) - min(A_sum, B_sum)
    print('%d :: E=%.2f, P=%.3f, A=%s, B=%s, Sum(A)=%d, Sum(B)=%d, OPT=%d' %
      (i, sol['E'], sol['P'], A, B, A_sum, B_sum, OPT))
```

3.2 Binary Integer Linear Programming

Background info:

- Linear Programming (LP), Integer Linear Programming(ILP), and Mixed Integer Linear Pro-

gramming (MILP)

- Binary Integer Linear Programming(BILP)

BILP and also ILP in general is a constrained optimization problem - we have to optimize an objective while satisfying specified constrains. BILP has numerous applications in many fields.

Formally, let us say $x_1, x_2, ...x_N$ are the binary variables in our BILP problem where $x_i \in \{0, 1\}, \forall i$. Our goal is to find the combinations of $x_1, x_2, ...x_N$ that produce optimal value of objective $c^T x$ while satisfying all the constraints in $Ax = b$. Here the matrix A and vector b holds all the coefficients of constraints. We already stated there are N binary variables, so $x \in \{0, 1\}^N$. If there are M constraints in any given problem, then $b \in R^M$ and $A \in R^{M \times N}$.

The objective $c^T x$ can be either to find maxima or minima. We like to solve minima problems since the quantum annealer finds minima or ground state of energy function. So, when needed, we shall convert maximization into minimization objective by negating all the coefficients (i.e., $c_i^{minimize} = -c_i^{maximize}, \forall i$) .

The constrains in standard form, $Ax = b$, has all equality constraints. In reality we might have mixture of equalities and inequalities (such as $<, \leq, >, \geq$). There are techniques to transform them into equalities (we will see them later, for now, let us work out the standard form).

Recall from the previous example that, we need to build an energy function E for annealer such that the combination of $x_1, x_2, ...x_N$ yields the minima or ground state. That is, the state yielding minima satisfies does not violate any constraint in $Ax = b$ and yields minima of $c^T x$.

$E = c^T x$ yields minima but doesnt consider constraints. So let us consider adding a term $(Ax-b)^2$, since it is 0 for valid answer and positive for any violations.

Therefore, $E = c^T x + (Ax - b)^2$ yields minima and accounts for constrains, but the issue is, optimizer might choose to violate some constrains when penalty from $(Ax - B)^2$ is smaller compared to objective gain in $c^T x$

So, to ensure no violation of any constraint, we need to scale the penalty such that penalty is bigger than the objective in the violated case.
i.e., $E = c^T x + p(Ax - b)^2$ where $p \geq \sum_i c_i$

Let us simplify this:

$$
\begin{aligned}
E &= c^T x + p(Ax - b)^2 \\
&= c^T x + p(x^T A^T Ax + b^T b - 2b^T Ax) \\
&\approx c^T x + p(x^T A^T Ax - 2b^T Ax) && \text{since } b^T b \text{ is same for all combinations} \\
&\approx [c^T - 2p * b^T A]x + p * x^T A^T Ax
\end{aligned}
$$

What we have here now is a vector $[c^T - 2p * b^T A]$ and a matrix $p * x^T A^T A$
Although it looks like Ising equation with linear and quadratic parts as in equation 1, but it is not, since the range of variables $x_i \in \{0, 1\}$. Perhaps this is close to QUBO (equation 2), except that we have a linear term $[c^T - 2p * b^T A]$. If we juxtapose QUBO and BILP, we see these are very related in a way that BILP has constrained optimization and QUBO is Unconstrained optimization. We formulated constrains of BILP with the extra quadratic term in QUBO.

We can further simplify it to obtain the QUBO form.

We know when $x \in \{0, 1\}$, $x^2 = x$
, $[c^T - 2p * b^T A]x = x^T diag(c^T - 2p * b^T A)x$ where $diag()$ produces a diagonal matrix from the input vector.

$$E \approx x^T [diag(c^T - 2p * b^T A) + p * A^T A]x$$

This equation is in QUBO form, and we know QUBO problem can be easily transformed to Ising problem. D-Wave SDK offers an API to do this for us (see in the below example).

Now that we have all the maths needed to embed a BILP problem to quantum graph, let us test it out with the Python API

```
from dwave_sapi2.local import local_connection
from dwave_sapi2.remote import RemoteConnection
from dwave_sapi2.core import solve_ising, solve_qubo
from dwave_sapi2.util import get_hardware_adjacency
from dwave_sapi2.embedding import find_embedding, embed_problem, unembed_answer
from dwave_sapi2.util import  qubo_to_ising
import numpy as np
from copy import copy

# Local solver is a simulated quantum annealer
print 'Available Local Solvers:', local_connection.solver_names()
loc_solver = local_connection.get_solver("c4-sw_optimize")

# Remote solver is the real quantum annealer, get URL and tokens of yours
# url = "https://<...>.isi.edu/sapi"
# token = ""
# rem_solver = RemoteConnection(url, token).get_solver("DW2X")

class QBILP(object):

  def __init__(self, solver, verbose=False):
    self.solver = solver
    self.verbose = verbose

  def make_ising(self, A, b, c, maximize=False):
    m, n = A.shape
    print("%d variables, %d constraints" % (n, m))
    assert c.shape[0] == n
    assert b.shape[0] == m

    if maximize: # reverse the signs
      c = copy(c)
      for i in range(n):
        c[i] = -1 * c[i]

    p = sum(abs(ci) for ci in c) # penalty

    AT_A = np.matmul(A.transpose(), A)
    bT_A = np.matmul(b.reshape(1, m), A)

    h = c.reshape(1, n) - 2 * p * bT_A
    Q =  p * AT_A
    h = h.flatten().tolist()
    for i, v in enumerate(h):
      Q[i,i] += v # add h to diagonally

    sparse_Q = {}
    for i in range(n):
      for j in range(n):
        if Q[i, j] != 0:
          sparse_Q[i, j] = Q[i,j]
    h, J, offset = qubo_to_ising(sparse_Q) # Transformation!
    return h, J

  def solve(self, A, b, c, maximize=False, num_reads=1000, params=None):

    h, J = self.make_ising(A, b, c, maximize)

    # Search for embeddings of J which uses only the active couplers
    hw_adj = get_hardware_adjacency(self.solver)
```

```
    embeddings = find_embedding(J, hw_adj, verbose=self.verbose)

    # Now embed the problem with whatever we found in previous step
    [h0, j0, jc, new_embeds] = embed_problem(h, J, embeddings, hw_adj)
    # j0 has updated coupler weights.
    # jc has cloning of qbits to represent a single logical variable. We need both
    emb_j = j0.copy()
    emb_j.update(jc)

    result = solve_ising(self.solver, h0, emb_j, num_reads=num_reads)
    sols = unembed_answer(result['solutions'], new_embeds, 'minimize_energy', h, J)
    res = []   # post process the result
    for i, sol in enumerate(sols):
      energy = result['energies'][i]
      prob = 1.0 * result['num_occurrences'][i] / num_reads
      # Ising to binary (undo mapping):  -1 -> 0 and +1 -> +1
      sol = [(i, 0) if bit == -1 else (i, 1) \
        for i, bit in enumerate(sol) if bit != 3]  # unused bits have value of 3
      res.append({'E': energy, 'P': prob, 'S': sol})
    return res
```

4 Limitations

Can any Ising Model problem be embedded to D-Wave's quantum hardware? No, that is not right. Surely there are limitations and I faced the following:

- If there are more variables than the qbits available in the hardware, then our problem doesn't fit in.
- The way qbits are connected can be seen as a graph on 2d plane. This graph is not fully connected, so not all $J_{i,j}$ are legal (Chimera topology). There are tricks to cope with this limitation at the expense of more quantum bits as we saw earlier, but that only reduces the size of problems that can be embedded to quantum graph.
- h and J has limited ranges: $[-2, +2]$. Sure, the out of range input can be standard normalized, but the values that has lot of variance will yield very small floating point numbers. The hardware support is limited to just 5 bit floating point precision (In 2017, D-Wave 2. This might change in the following years). I wish they support 32 bit single precision (if not 64 bit doubles) like the classical CPU ALU chips, but that seems to be too much to ask.

5 Conclusion

I hope this tutorial uncovered some of the mysteries of quantum optimization (QO) for beginners. Quantum annealer is relatively easier to use once we figure out how to express optimization problems in either Ising or QUBO forms. In this early stages, there are a few practical limitations, but I hope they will be resolved as the field matures.