Driving to the 48 USA State Capitals:
Programming the D-Wave QPU

North Carolina State University
Joel M. Gottlieb
February 6, 2018
Overview

• *Review* (*Scott Pakin and D-Wave*)
• qbsolv
• 7-city Traveling Salesman Problem
• 48-city Traveling Salesman Problem
• Summary
## Review: Programming Model

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUBIT</td>
<td>Quantum bit which participates in annealing cycle and settles into one of two possible final states:</td>
</tr>
<tr>
<td>COUPLER</td>
<td>Physical device that allows one qubit to influence another qubit</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>Real-valued constant associated with each qubit, which influences the qubit’s tendency to collapse into its two possible final states; controlled by the programmer</td>
</tr>
<tr>
<td>STRENGTH</td>
<td>Real-valued constant associated with each coupler, which controls the influence exerted by one qubit on another; controlled by the programmer</td>
</tr>
<tr>
<td>OBJECTIVE</td>
<td>Real-valued function which is minimized during the annealing cycle</td>
</tr>
</tbody>
</table>

The system samples from the that minimize the objective
Review: The overall process

• Need to map problems into binary variables

• Need to map the binary variable expressions into linear terms and pair quadratic terms
  – QUBO: Quadratic Unconstrained Binary Optimization

• Run the QUBO on the D-Wave QPU
• Interpret the results
• Revisit the mapping...
QUBOs can be found in many fields

• Optimization
  – Recent: financial portfolio management
  – Recent: hospital re-admission statistics (Medicare funding)
  – Recent: bioinformatics / Multiple Sequence Alignment
  – Recent: nuclear power plant failure scenario analysis
  – Mathematical problems like Minimum Vertex Cover
  – Job-shop scheduling, other graph problems (Map Coloring, vertex set color)

• Sampling (from probability distribution)
Application example flow

- Start with binary variables in problem domain
- Convert to QUBO
- Solve QUBO with qbsolv
- Convert bit vector back to variables in problem domain
- Interpret qbsolv results and adjust accordingly
Overview

• Review (from Scott Pakin and D-Wave)
• $qbsolv$
• 7-city Traveling Salesman Problem
• 48-city Traveling Salesman Problem
• Summary
The need for qbsolv

• Many problems require many more qubits or couplers than are available with the current chip.

• Examples:
  – Portfolio Optimization example requires 63-variable complete graph, but 2000Q does not go beyond ~50-qubit complete graph with direct embedding
  – 48-city Traveling Salesman requires complete graph, and it might be possible on the 2000Q, but the needed embedding would have very long chains, which do not perform well
  – Traffic flow optimization example using 418 cars (with 3 routes each) require almost 2000 highly-connected variables, too many to fit directly on the 2000Q chip.
qbsolv

- Hybrid quantum/classical QUBO solver (tabu = classical heuristic solver)
- Designed for problems too large and/or too dense to run on D-Wave quantum computer
- Divides problems into chunks, and iterates on sub-QUBOs (similar to HFS algorithm)
- Open-source: [https://github.com/dwavesystems/qbsolv](https://github.com/dwavesystems/qbsolv)
- Can be used standalone, or with 128-qubit simulator, or with QPU
- Produces a single bitstring solution representing the final states of all the binary variables
Motivating Algorithm

• "A Multilevel Algorithm for Large Unconstrained Binary Quadratic Optimization", Wang, Lu, Glover, and Hao [2012]

• Principles
  – Identify the *backbone* of the QUBO; *i.e.*, the variable settings that are correlated for all valid answers, or, contribute the most to a local optimum
  – Select subQUBOs by most/least impact of each variable in determining the optimum
  – Solve the subQUBOs with a solver known to run effectively at smaller scale
  – Propagate subQUBO answer out to original variables in full QUBO
  – Iterate above steps until no further improvement

Copyright © D-Wave Systems Inc.
How qbsolv works

• Hybrid algorithm:
  
  – Identify the significant rows and columns of the larger problem (What changes a lot with spin flips? What doesn’t?)
  
  – Create a smaller representative QUBO of that subset
  
  – Execute that smaller QUBO on the D-Wave system (pre-computed embedding, speeds up run-time)
  
  – Use the answer to guide the larger solver (new starting point, closer to the minimum)
Example qbsolv Output

$ qbsolv -i bpp50.qubo1.qubo

50 Number of bits in solution

1011111110100111110100110111111101111111110110110

-5176.00000 Energy of solution

0 Number of Partitioned calls

0.21352 seconds of classic cpu time
Overview

• Review (from Scott Pakin and D-Wave)
• qbsolv
• 7-city Traveling Salesman Problem
• 48-city Traveling Salesman Problem
• Summary
7-city Traveling Salesman Problem

• A learning exercise

• We will explore:
  1. QUBO for leg variables
  2. City visit constraints
  3. Embedding and chain constraints
  4. Importance of parameters and problem construction
Problem Specifications

Optimization:

• Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

• Given a length $L$, decide whether the list of cities and distances has any tour shorter than $L$.

• Symmetric TSP; Undirected graph, distances in miles; distances obtained from various Web sites.
The Cities

A = Albuquerque, NM
B = Boston, MA
C = Charlotte, NC
D = Detroit, MI
E = Evanston, IL
F = Frankfort, KY
G = Gulfport, MS

(Distances found in file tsp7.b)

Example path:
Path = (A->B) + (B->C) + (C->D) + (D->E) + (E->F) + (F->G) + (G->A)
QUBO: Leg Variables

Binary Variable \( \text{ab} \):

1 if the trip includes the segment A -> B
0 if the trip does not include A->B

Distances between cities A and B denoted by \( D_{ab} \)

Distance to Minimize:

\[ D_{ab} \cdot \text{ab} + D_{ac} \cdot \text{ac} + D_{ad} \cdot \text{ad} + \ldots + f_{fg} \cdot D_{fg} \]

How do we convert this into a QUBO?
Converting into a QUBO

Each city must be visited exactly twice – once arriving, and once departing.

For city A, we must have:

$$ab + ac + ad + ae + af + ag - 2 = 0$$

For City B,

$$ab + bc + bd + be + bf + bg - 2 = 0$$

And so on, up to city G.

The QUBO to minimize:

$$D_{ab} \ast ab + D_{ac} \ast ac + D_{ad} \ast ad + \ldots + f_{g} \ast D_{fg} + \gamma ((ab + ac + ad + \ldots - 2)^{2} + (ab + bc + bd + be + bf + bg - 2)^{2} + \ldots)$$
Algebraic results: expanding the equation

- Groups of terms of the form:
  - \( 3 \cdot vertex \cdot ab \): favor visiting the path A->B

- Groups of terms of the form:
  - \( 2 \cdot vertex \cdot ab \cdot ag \): penalize visiting the paths A->B and A->G
    (some of these will be selected)

- There are seven explicit \texttt{dw} assert statements (which help identify valid solutions):
  \[
  \text{assert:vertex:} ab + ac + ad + ae + af + ag - 2
  \]

- \textit{vertex} is a Lagrangian multiplier; needs to be weighed against the inter-city distances
Steps to run the 7-city TSP (write a run script)

- Prepared `tsp7.b` (parameter file) and `tsp7.q` (QUBO file) by hand
- Embed the problem onto the 128-qubit simulator using the `dw embed` command (the default embedder algorithm tries to find a way to map the needed logical qubits onto available physical qubits)
- Try a value of `vertex` and a value of `param_chain`, another adjustable parameter (controls the chain strength in the embedding)
- Run `dw bind` to bind the parameter values in the B file to the QUBO
- Run `dw exec` to run the problem on the simulator (or QPU)
- Run `dw val` to interpret and validate the output
Parameter explorations

- Earlier, we mentioned `vertex` and `param_chain`, the adjustable parameters.
- `Vertex` controls the strength of obeying the constraints.
- `Param_chain` controls the strength of chains in the embedding.
- When running the problem, parameter space must be explored, to find largest possible number of solutions.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>1300</th>
<th>1400</th>
<th>1500</th>
</tr>
</thead>
<tbody>
<tr>
<td>4500</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4750</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Looking at the lowest-energy solution

- `dw val --s 1 tsp7.sol` produces output that looks like this:

  **** SOLUTION 1 ****
  ab <= 0
  af <= 1
  ag <= 1
  bc <= 1
  ...
  VALID: Y
  SAMPLES: 4
  OBJECTIVE: -33778.00

And we get the path:

af + ef + de + bd + bc + cg + ag
a -> f -> e -> d -> b -> c -> g -> a
A map of the lowest-energy solution
Interpreting the energy

The path: \( af + ef + de + bd + bc + cg + ag \)

Adding up the miles: 5422 miles

\( dw \) tells us the energy is -33778

For each vertex, there will be two nonzero terms of the form:

-3 * vertex * \( af \) = -3 * vertex

And one term of the form:

2 * vertex * \( af \) * \( ag \) = 2 * vertex

Thus: -4 * vertex per 7 cities => -28 * vertex

Energy = Total Mileage – 28 * vertex

(for vertex = 1400, this equality works)
An interesting issue

A valid solution and different path:

\[ ae \ ag \ bc \ bd \ cf \ df \ eg \]

Notice: \( A \rightarrow E \rightarrow G \rightarrow A \)

This is called a subloop: the path split into a 3-loop and a 4-loop

- Subloops are not prevented by “visit twice” constraint
- Subloops are not desirable solutions; we would have to write many more constraints to eliminate them

• The Lucas formulation (Permutation matrix) rules out subloops (next section)
Conclusions from 7-city TSP

- **dw** can be used as part of overall toolbox, start to finish

- Good problem for exploring Lagrangian parameters, understanding solutions, how QUBO construction affects solutions (subloop problem).

- Good problem for understanding chains and embedding

- Has quick physical interpretation of solution

- Good problem for understanding what the QPU does, what its inputs are, and what it returns
Overview

• Review (from Scott Pakin and D-Wave)
• qbsolv
• 7-city Traveling Salesman Problem
• 48-city Traveling Salesman Problem
• Summary
48-city Problem Specifications

- Symmetric TSP; undirected graph; distances in miles

- Driving distances obtained from Google Maps

- We can reduce the complexity of the problem by 1- assume last city visited is alphabetically last city (Cheyenne, WY)

First approach: QUBO (ref. Andrew Lucas)

QUBO approach means building a permutation matrix of 0’s and 1’s and then introducing quadratic terms to include distances:

Tour represented: city 4 \rightarrow city 2 \rightarrow city 1 \rightarrow city 6 \rightarrow city 3 \rightarrow city 5
QUBO (continued)

For each row and column, we introduce a constraint via the following QUBO terms:

\[
Constraint = A \left( -1 + \sum_{i=1}^{N} x_{1,i} \right)^2
\]

For row 1

\[
Constraint = A \left( -1 + \sum_{j=1}^{N} x_{j,1} \right)^2
\]

For column 1

Distance from the first city visited to the second city visited is computed like this:

\[
Distance = B \sum_{i=1}^{N} \sum_{j=1}^{N} d_{i,j} x_{i,1} x_{j,2}
\]

From 1st to 2nd city
Algebraic results

• QUBO approach requires boolean variables to encode a tour, but “last city visited” assumption provides reduction

• Constant term (2N-2)A: one for each row + column, reduced

• Each variable gets a term -2A multiplied by it (e.g. -2A) to incentivize visiting it on a particular step (e.g., city 1, step i)

• Pairwise terms penalize cities being visited twice, or visited on same step (e.g. 2A or 2A)

• Distance terms penalize following an edge between two cities (the distance term raises the overall energy) (e.g. B)

• ”Last city visited” assumption leads to diagonal terms B and B (last and first have to get to Cheyenne)
Python program generate_qubo.py

- Number of cities N is input, cannot go beyond 48, but easy to extend
- Divided through by B, so that there is only one adjustable parameter A
- Read in the inter-city distances from state_capitals.txt, and create distance matrix
- Diagonal terms will be -2A, except when we need to include “last city visited” reduction effect
- qbsolv requires off-diagonal terms to have i < j
- Use QUBO_details library to write QUBO
Looking at the 48-city QUBO

- For $N = 48$, with the reduction, $(N-1)^2$ Boolean variables (2209)
- A set to 8500 (required: larger than biggest inter-city distance)
- Most of the diagonals are -17000 ($-2 \times A$), but some have the distance added in, to the last city (“How far from here to Cheyenne?”)
- Many of the off-diagonal terms are 17000 ($2 \times A$) as well
- The last off-diagonal term is $(2207,2208)$, as we expect
Python program interpret_lucas.py

- Read in the inter-city distances from state_capitals.txt, and create distance matrix
- Read in the names of the cities
- Read in the qbsolv output
- Associate the result bitstring with binary variables to compute the city tour
- Output a city tour for eventual drawing on Google Maps
- Useful for debugging QUBO/energy
Python program generate_map.py

- Read in the city tour from the previous step
- Read in an HTML template for generating Google USA map
- Insert the city tour into the HTML template
- Write out an HTML page which can be displayed, containing the Google map of the route!
The solution – meh
A possible “Optimal solution” from Randal Olson

• Randal Olson from the University of Pennsylvania wanted to drive all 48 state capitol buildings
• Genetic algorithm
• Olson’s Web page indicates his route is 13,310 miles, but he focuses on state capitol buildings
• http://www.randalolson.com/2016/06/05/computing-optimal-road-trips-on-a-limited-budget/
Randal Olson’s solution – way better
Insights

• Why didn’t we get the better solution?
• Notice that the difference between “good” and “bad” solutions is less than 1% of the overall energy – why is this?
• Some problems are mathematically not great for the D-Wave
• Only $N-1=47$ 1’s can be turned on; the qbsolv algorithms are more effective with larger numbers of bitflips (I need to try this the other way)
• For your problems, focus initially on special cases which can be intuitively understood (TSP with 4 cities, etc.)
• Use post-qbsolv code to test solutions for consistency (for example, introduce additional parameters and establish that the solutions don’t change)
Overview

• Review (from Scott Pakin and D-Wave)
• qbsolv
• 7-city Traveling Salesman Problem
• 48-city Traveling Salesman Problem
• Summary
Summary

• Hybrid classical/quantum approach to using D-Wave
• “Toolbox” approach: Python, dw, qbsolv, simulator, (hardware)
• Problem has understandable graphical representation
• “This problem is not particularly large, but hard.”
• The goal: you can formulate your client’s problems into QUBO and then run them, even if solutions are not immediately optimal
• The post-qbsolv code is vital for at least three reasons:
  – Interpreting the bitstrings and understanding the solutions
  – Debugging the math of converting the problem to QUBO
  – Debugging the code representing the QUBO