Quantum Uncertainty Quantification for Physical Models using ToQ.jl

Daniel O’Malley & Velimir V. Vesselinov

Computational Earth Science
Los Alamos National Laboratory

Unclassified: LA-UR-16-27674
Programming D-Wave machines

\[ \hat{q} = \min_{q \in \{0,1\}^n} \sum_{i=1}^{n} a_i q_i + \sum_{i=1}^{n} \sum_{j=1}^{i-1} b_{ij} q_i q_j \]

\[ q \sim e^{-\beta \left( \sum_{i=1}^{n} a_i q_i + \sum_{i=1}^{n} \sum_{j=1}^{i-1} b_{ij} q_i q_j \right)} \]

- The core task of programming a D-Wave machine is to assemble

\[ \sum_{i=1}^{n} a_i q_i + \sum_{i=1}^{n} \sum_{j=1}^{i-1} b_{ij} q_i q_j \]

in such a way that one of the above approximations will be useful
How can ToQ.jl help?

- ToQ.jl makes it relatively easy to assemble a “logical QUBO”
  - Don’t have to worry about the sparsity of $b_{ij}$
  - Don’t have to worry about the bounds on $a_i$ and $b_{ij}$
  - Do have all the features of Julia
- ToQ.jl makes it very easy to use different backends
  - dw
  - Python SAPI
  - qbsolv
- ToQ.jl makes it very easy to write hybrid classical/quantum programs
  - Use Julia to write the classical part
  - Use Julia+ToQ.jl to interact with the D-Wave
- Sidenote: ToQ.jl is a misnomer and has nothing to do with D-Wave’s ToQ – name change coming soon
Assembling a logical QUBO
Coloring the map of Canada

- Color the 13 Canadian provinces with 3 colors
- Assign exactly one of three colors to each province
- No two neighboring provinces can have the same color
Assembling a logical QUBO
Coloring the map of Canada

- Color the 13 Canadian provinces with 3 colors
- Assign exactly one of three colors to each province
- No two neighboring provinces can have the same color

provinces = ["BC", "YK", "NW", "AB", ...]
neighbors = Dict()
neighbors["BC"] = ["YK", "NW", "AB"]
neighbors["YK"] = ["BC", "NW"]
neighbors["NW"] = ["YK", "BC", "AB", "SK", "NV"]
neighbors["AB"] = ["BC", "NW", "SK"]
neighbors["SK"] = ["AB", "NW", "MT"]
neighbors["NV"] = ["NW", "MT"]
neighbors["MT"] = ["NV", "SK", "ON"]
neighbors["ON"] = ["MT", "QB"]
neighbors["QB"] = ["ON", "NB", "NL"]
neighbors["NB"] = ["QB", "NS"]
neighbors["NS"] = ["NB"]
neighbors["PE"] = []
neighbors["NL"] = ["QB"]
Assembling a logical QUBO
Coloring the map of Canada

- Color the 13 Canadian provinces with 3 colors
- Assign exactly one of three colors to each province
- No two neighboring provinces can have the same color

\[
m = \text{ToQ.Model("canada_model", "laptop", "c4-sw_sample", "workingdir", "c4")}
\]

\[
@\text{defvar m red[1:length(provinces)]}
@\text{defvar m green[1:length(provinces)]}
@\text{defvar m blue[1:length(provinces)]}
\]
Assembling a logical QUBO
Coloring the map of Canada

- Color the 13 Canadian provinces with 3 colors
- Assign exactly one of three colors to each province
- No two neighboring provinces can have the same color

\[-(q_1 + q_2 + q_3) + 2(q_1q_2 + q_1q_3 + q_2q_3)\]

for \(i = 1:\text{length}(\text{provinces})\)
\[
\begin{align*}
@\text{addterm} & \text{ m } -1 \ast \text{ red}[i] \\
@\text{addterm} & \text{ m } -1 \ast \text{ green}[i] \\
@\text{addterm} & \text{ m } -1 \ast \text{ blue}[i] \\
@\text{addterm} & \text{ m } 2 \ast \text{ red}[i] \ast \text{ green}[i] \\
@\text{addterm} & \text{ m } 2 \ast \text{ red}[i] \ast \text{ blue}[i] \\
@\text{addterm} & \text{ m } 2 \ast \text{ green}[i] \ast \text{ blue}[i]
\end{align*}
\]
end
Assembling a logical QUBO
Coloring the map of Canada

- Color the 13 Canadian provinces with 3 colors
  \[ r_1r_2 + g_1g_2 + b_1b_2 \]
  
  for \( j = 1:\text{length(provinces)} \)
  for \( k = 1:j-1 \)
    if provinces[k] in neighbors[provinces[j]]
      @addterm m red[j] * red[k]
      @addterm m green[j] * green[k]
      @addterm m blue[j] * blue[k]
    end
  end
end

- Assign exactly one of three colors to each province

- No two neighboring provinces can have the same color
I’ve got the QUBO, now what?
Coloring the map of Canada

#sample with dw
ToQ.solve!(m; numreads=100, param_chain=2)
#load the first dw sample
@loadsolution m energy occurrences isvalid 1
#print the first dw sample
println(hcat(red.value, blue.value, green.value))

#sample with Python SAPI
ToQ.solvesapi!(m; num_reads=100, param_chain=2)
#load the first SAPI sample
@loadsolution m energy occurrences isvalid 1
#print the first SAPI sample
println(hcat(red.value, blue.value, green.value))

#solve with qbsolv
ToQ.qbsolv!(m; minval=-13 * w1)
#print the qbsolv solution
println(hcat(red.value, blue.value, green.value))
function colormap(regions, neighbors, numcolors; w1=1, w2=1)
    colornames = ["red", "orange", "yellow", "green", "blue", "indigo", "violet"]
    m = ToQ.Model("mapcolor", "laptop", "c4-sw_sample", "workingdir", "c4")
    @defvar m colors[1:length(regions), 1:numcolors]
    for i = 1:length(regions)
        for j = 1:numcolors
            @addterm m -w1 * colors[i, j]
            for k = 1:j - 1
                @addterm m 2 * w1 * colors[i, j] * colors[i, k]
            end
        end
    end
    for j = 1:length(regions)
        for k = 1:j - 1
            if regions[k] in neighbors[regions[j]]
                for i = 1:numcolors
                    @addterm m w2 * colors[j, i] * colors[k, i]
                end
            end
        end
    end
    ToQ.qbsolv!(m; minval=-length(regions) * w1)
    result = Dict(zip(regions, map(i->colornames[indmax(vec(colors.value[i, :]))], 1:length(regions))))
    return result
end

counties, county_neighbors = parseusa("county_adjacency.txt")
usa_answer = colormap(counties, county_neighbors, 4)
The developed UQ methods are not “ready for prime time”

However, there are clear paths for improvement that could be pursued with more time & effort
The physics problem

\[ \nabla \cdot (k \nabla u) = f \]
The physics problem

\[ \nabla \cdot (k \nabla u) = f \]
What we know

\[ \nabla \cdot (k \nabla u) = f \]
What we don’t know

\[ \nabla \cdot (k \nabla u) = f \]
The uncertainty quantification

Two sets of bits that will be represented on the D-Wave hardware: the $q_i$ bits are used to represent $k$ and the $r_i$ bits are used to represent $f$.

$$u_i - \hat{u}_i \sim N(0, 1)$$
$$k_i = k_{low} + q_i (k_{high} - k_{low})$$
$$f_i = -r_i$$
$$0 = k_{i-1} u_{i-1} - (k_{i-1} + k_i) u_i + k_i u_{i+1} - f_i$$

$$H(q, r) = \sum_{i=2}^{N-1} [k_{i-1} u_{i-1} - (k_{i-1} + k_i) u_i + k_i u_{i+1} - f_i]^2$$ (1)

Get samples from the D-Wave with likelihood

$$\exp(-\beta H^*(q, r))$$

use importance sampling to estimate statistics of $k$ and $f$. 

1^{st} \text{ order approximation}

\[ q \sim e^{-\beta \left( \sum_{i=1}^{n} a_i q_i + \sum_{i=1}^{n} \sum_{j=1}^{i-1} b_{ij} q_i q_j \right)} \]

Let \( a_i = h \ \forall i \) and \( b_{ij} = 0 \ \forall i, j \)

\[ P(q = 1; h, \beta) = \frac{\exp(-\beta h)}{\exp(-\beta h) + 1} \]

\[ \hat{\beta} = \arg \min_{\beta} \sum_{i=1}^{N} (P(q = 1; h_i, \beta) - \sum_{j=1}^{M} q_j^i / M)^2 \]
Importance sampling: Boltzmann $\rightarrow$ Target

Boltzmann

\[ e^{-\beta \left( \sum_{i=1}^{n} a_i q_i + \sum_{i=1}^{n} \sum_{j=1}^{i-1} b_{ij} q_i q_j \right) } \]

Target

\[ \frac{f(q)}{e^{-\beta \left( \sum_{i=1}^{n} a_i q_i + \sum_{i=1}^{n} \sum_{j=1}^{i-1} b_{ij} q_i q_j \right) }} \]
Possible next steps

- Use a better approximation for the D-Wave sampling likelihood
  - Results that are being presented today?
  - \[ \exp \left\{ -\beta \left[ \sum_i (1 + \epsilon_i) a_i q_i + \sum_{i<j} (1 + \epsilon_{ij}) b_{ij} q_i q_j \right] \right\} \]
  - Better Boltzmann sampling from D-Wave
  - Finish off each sample with a short MCMC chain
- Reduce the impact of embedding
  - D-Wave’s “product X”
Conclusion

- Importance sampling did not work well here, but we can and (someone) will do better in the future
- ToQ.jl is available on GitHub
  - https://github.com/losalamos/ToQ.jl
- Many thanks to the D-Wave team for their help along the way