D:wave
The Quantum Computing Company™

WHITEPAPER

Programming with D-Wave: Map Coloring Problem

E. D. Dahl, D-Wave Systems
November 2013

Overview

Quantum computing, as implemented in the D-Wave system, is described by a simple but largely unfamiliar programming model. Using a simple map coloring problem this white paper describes the entire set of transformations needed to find solutions by executing a single quantum machine instruction (QMI) within this programming model. This “direct embedding” is one of several ways to program the D-Wave quantum computer.

CONTACT

Corporate Headquarters
3033 Beta Avenue
Burnaby, British Columbia V5G 4M9
Canada
Tel. 604-630-1428

US Office
2650 E. Bay Shore Road
Palo Alto, CA

Email: info@dwavesys.com

www.dwavesys.com

1. Introduction

Quantum computers utilize quantum bits (qubits) to hold information. The behavior of each qubit is governed by the laws of quantum mechanics, enabling qubits to be in a "superposition" state – that is, both a 0 and a 1 at the same time, until an outside event causes it to "collapse" into either a 0 or a 1. This property is foreign to our everyday experiences in the macroscopic world, but it is the basis upon which a quantum computer is constructed. Exploiting this property gives a quantum computer the ability to quickly solve certain classes of complex problems such as optimization, machine learning and sampling problems.

Programming a quantum computer is very different than programming a traditional computer. To program the system a user maps a problem into a search for the "lowest point in a vast landscape" which corresponds to the best possible outcome. The processor considers all the possibilities simultaneously to determine the lowest energy required to form those relationships. Because a quantum computer is probabilistic rather than deterministic, the computer returns many very good answers in a short amount of time - 10,000 answers in one second. This gives the user not only the optimal solution or a single answer, but also other alternatives to choose from.

This paper describes the set of transformations to turn a map coloring problem into a single quantum machine instruction (QMI) using the "direct embedding" programming model, one of a few different methods to program a D-Wave system. Map coloring represents a large class of combinatorial optimization problems and is thus a good model problem for the D-Wave system. While finding a valid coloring of the map of Canada is not a hard exercise (even by hand), our focus is on the translation from the problem to the programming model.

Maps of Canada display ten provinces and three territories. Typical maps (see figure 1) assign a color to each of the thirteen regions subject to a simple constraint: two regions that share a border receive different colors. (Regions touching at one or more isolated points, for example Nunavut and Saskatchewan, are not considered to share a border.) The point of such a coloring is obvious: it allows our eyes to easily distinguish geographic areas that belong to a given region.

Figure 1: Map of Canada's ten provinces and three territories

We begin by defining the programming model for the D-Wave system, introducing important concepts such as *qubits, couplers, weights and strengths*. These entities appear in each QMI - either implicitly or explicitly. Next, we choose a correspondence or encoding between the colors for a region and the qubit values. Once the encoding is fixed, we work out the form of a QMI which will give valid colorings. This task is broken into four simple steps:

1. Turn on one of several qubits
2. Map a single region to a unit cell
3. Implement neighbor constraints using intercell couplers
4. Clone neighbors to meet all neighbor constraints

These steps suffice for a map which is simple enough to fit entirely into the D-Wave system. This paper does not address implementation aspects of the D-Wave system. In the same way that a discussion of the architecture of a modern microprocessor does not include discussions of transistors or digressions into aspects of solid-state physics, we avoid describing superconducting quantum interference devices and Josephson junctions. The programming model presented next is what one needs to know so that one can program the system effectively.

2. Programming Model

Classical computers are composed of registers and memory, the collective contents of which are referred to as state. Instructions for a classical computer act deterministically on this state, transforming the contents of a few registers or memory locations at a time. The D-Wave quantum computer differs fundamentally from this description of a classical computer in two ways. First, it has no registers or memory locations and hence does not possess anything which is analogous to state. Second, as noted before, instructions for the D-Wave quantum computer are not deterministic, but return probabilistic results.

If the D-Wave quantum computer has no registers or memory locations, a natural question arises: how do we learn anything from having executed a quantum machine instruction? The answer is that we are given samples from a distribution, as a side effect of executing the QMI. To explain what these samples are and how this distribution is defined, we must first introduce several entities which are key to our programming model.

The first such entity is the *qubit*, which is simply a variable (e.g., q) that takes values from the set {0, 1}. The D-Wave system has many qubits, so we will often use a subscript to label our qubits: q_i. The programming model does not allow the programmer to directly set the value of these qubits. Instead, we *influence* the qubits and the D-Wave system responds to the influences we provide.

How do we influence qubits? There are two ways. First, associated to each qubit is a *weight* which is part of each QMI and is under programmer control. We denote the weight associated to qubit q_i by the symbol a_i.

The second way relies on the notion of a *coupler*, which allows us to control the influence that one qubit exerts on a second. A coupler is always identified with exactly two qubits, and so we say that the coupler *connects* the qubits. Just as a weight is associated with each qubit, we associate a *strength* with each coupler. If a coupler connects qubits q_i and q_j, denote the strength of the coupler by b_{ij}.

Now that we've defined qubits and weights and couplers and strengths, we can write down an objective function that will define the distribution from which our samples will be selected. This objective function takes as its input the values of the weights (a_i), the strengths (b_{ij}) and the qubits (q_i). For each specification of its input, it returns an objective value, or sometimes more simply an *objective*:

$$O(\mathbf{a}, \mathbf{b}; \mathbf{q}) = \sum_{i=1}^{N} a_i q_i + \sum_{<i,j>} b_{ij} q_i q_j$$

The first summation in the objective function encompasses all qubits and so we put limits from 1 to N on the sum, where N is the number of qubits in our system. The second summation is over all couplers and we denote the pairs of qubits corresponding to a coupler by the angle bracket notation $< i, j >$. An important point to note is that each quantum machine instruction consists of exactly the a_i and b_{ij} values that appear in the above definition of the objective function.

Now that we have defined the objective function, we can describe the samples and the distribution from which they are drawn. Each sample is simply the collection of q_i values for the entire set of qubits which enter into our problem. The distribution is an equal weighting across all the samples that give the minimum value of our objective function.[1]

As a programmer, it is our job is to encode the various possible solutions to an optimization problem in the qubit variables q_i. Then we translate the constraints in our optimization problem into values of the weights a_i and strengths b_{ij} such that *when the objective is minimized* the qubits q_i will satisfy the constraints. In the next section we carry out each of these steps for our chosen example problem.

3 Map Coloring to QMI

Let C be the number of colors. Use a unary encoding of the possible colors for each region, assigning C qubits to each region of the map. If the **i**th color is assigned to a region, then the **i**th qubit associated with that region will have the value 1 in our samples and the other $C - 1$ qubits associated with that region will have the value 0. This encoding dictates the four steps required to build our objective function.

3.1 Turn on one of C qubits

First solve the simpler problem of turning on exactly one of two qubits. For a two qubit system, the objective is:

$$O(\mathbf{a}, \mathbf{b}; \mathbf{q}) = a_1 q_1 + a_2 q_2 + b_{12} q_1 q_2$$

Enumerate the four possible states in our distribution (see table 1). Our task is to choose the a_1, a_2 and b_{12} values so that our distribution consists of the state with $q_1 = 0$ and $q_2 = 1$ and also the state with $q_1 = 1$ and $q_2 = 0$. The other two states in which both q_i are equal to 0 or both q_i are equal to 1 should not be present in our distribution.

q_1	q_2	$O(\mathbf{a}, \mathbf{b}; \mathbf{q})$
0	0	0
0	**1**	a_2
1	0	a_1
1	**1**	$a_1 + a_2 + b_{12}$

Table 1: Four states of a two qubit system, plus objective

[1] In reality some fraction of samples may not correspond to the absolute minimum of the objective, but to values which are slightly above the minimum.

To do this choose a_1 and a_2 to be equal, so that the encodings of either color are equally likely in the distribution. Also choose these values to be less than 0, so the state characterized by $q_1 = 0$ and $q_2 = 0$ will not appear:

$$a_1 = a_2 < 0$$

To eliminate the state with $q_1 = 1$ and $q_2 = 1$ from our distribution, we can require:

$$a_1 + a_2 + b_{12} = 0$$

A solution to these equations is:[2]

$$a_1 = -1 \quad a_2 = -1 \quad b_{12} = 2$$

Substituting these values for a_1, a_2 and b_{12} into table 1 shows that the objective value is minimized for the two states in which exactly one of the q_i values is 1 and the other is 0. This means that the samples retrieved from the distribution generated by the QMI will consist solely of these two states.

These coefficient values would suffice if we only had two colors for our map regions, but most maps require more colors and so we must generalize to the case where there are C qubits representing the possible colors assigned to a region. We must find the values of the a_i and b_{ij} coefficients that will result in a distribution over those samples that have exactly one qubit turned on (i.e., equal to 1) and the other $C - 1$ qubits turned off (i.e., equal to 0).

To solve this problem, take a clue from the two-qubit problem solved above. In that problem we wanted the two states with exactly one qubit turned on to be equally represented in our distribution and thus we set *a1 = a2.* Our solution is *symmetric* if we interchange the two qubits, as we expect it to be. Apply this same principle to the case where we have three colors and a corresponding number of qubits in order to simplify the constraints.

If $C = 3$ then there are three qubits and the corresponding objective function is:

$$O(\mathbf{a}, \mathbf{b} ; \mathbf{q}) = a_1q_1 + a_2q_2 + a_3q_3 + b_{12}q_1q_2 + b_{13}q_1q_3 + b_{23}q_2q_3$$

Simplify this objective by applying the insight about the symmetry of the solutions, and require the three a_i values to equal one common value a. Similarly require the three b_{ij} values to equal one common value b:

$$O(\mathbf{a}, \mathbf{b} ; \mathbf{q}) = a(q_1 + q_2 + q_3) + b(q_1q_2 + q_1q_3 + q_2q_3)$$

q_1	q_2	q_3	objective
0	0	0	0
0	0	**1**	*a*
0	**1**	0	*a*
0	**1**	**1**	*2a + b*
1	0	0	*a*
1	0	**1**	*2a + b*
1	**1**	0	*2a + b*
1	**1**	**1**	*3a + 3b*

Table 2: Eight states of a three qubit system, plus objective

Tabulate the eight states of this system (see table 2). Taking a hint from our previous example, observe that setting $a = -1$ and $b = 2$ will cause the three states with exactly one qubit turned on to have objective value –1. Of the other five states, four will have objective value equal to 0 and one will have its objective equal to 3. This guarantees that our samples will consist only of those qubit patterns in which one of the three qubits is equal to 1.

A quick check confirms that this same symmetry argument can be applied to problems with $C = 4$ or higher. In all these cases, we can influence the distribution to contain only those qubit patterns with exactly one qubit turned on by choosing $a_i = -1$ for all the weights and $b_{ij} = 2$ for all the strengths.

3.2 Map logical qubits to physical qubits

To finish the first step of transforming the map coloring problem to a QMI, we introduced C qubits for each region in our map and initialized weights and strengths for the qubits and couplers. The pattern of connectivity of the qubits and couplers is represented in figure 2.

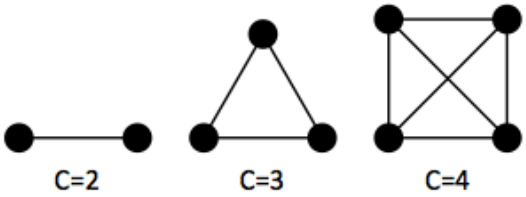

Figure 2: Complete graphs on two, three and four vertices

In this figure, each graph vertex represents a qubit and each edge represents a coupler between qubits. Figure 3 depicts a small portion of the pattern of physical qubits and couplers present in the D-Wave system corresponding to the unit cell. It is trivial to find many instances of the complete graph on two vertices within the D-Wave system, but there are no instances of the complete graph on three or four (or more) vertices. This poses our next challenge.

To solve this problem, introduce a distinction between the logical qubits and couplers discussed in the preceding section and the physical qubits and couplers of the real system. Each logical qubit corresponds (via an *embedding*) to one or more connected physical qubits, which we call a *chain*. To implement a coupler between logical qubits, it is enough to find a physical coupler connecting some physical qubit in the chain for the first logical qubit to another physical qubit in the chain for the second logical qubit.

This strategy allows us to easily map the complete graphs on up to four vertices to the unit cell of the D-Wave system. The chain for the first logical qubit corresponds to the top two physical qubits in the D-Wave unit cell. Likewise, the chain for the n^{th} logical qubit corresponds to the two physical qubits in the n^{th} row of the D-Wave unit cell. It is easy to confirm that there exists a physical coupler between each pair of chains, yielding a unit cell embedding for complete graphs on up to four vertices.

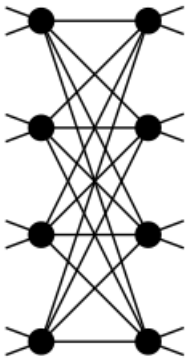

Figure 3: D-Wave unit cell (8 qubits, 16 couplers)

So that the physical qubits within one chain faithfully represent a single logical qubit, we must introduce weights and strengths for the physical qubits within a chain that cause them to stay aligned. By referring to the table of objective values for the two-qubit system, it is easy to see that assigning $a_1 = 1$, $a_2 = 1$ and $b_{12} = -2$ results in the aligned chains having objective 0 and the misaligned

chains having objective 1. Note that the weight and strengths necessary to implement this desired set of states are the negative of the weights and strengths used to solve the "1 of 2" problem. To complete this step, we must also supply a rule to map the weights and strengths for the logical qubits and couplers to the physical qubits and couplers that represent them. Each logical qubit is represented by a chain of length two, so we divide the weight of -1 for the logical qubit in half and apply a weight of -1/2 to each physical qubit in the chain. Likewise, we specified a strength of 2 for the logical couplers. Since the chains for each pair of logical qubits are connected by two physical couplers, we also divide the logical strength of 2 into two physical strengths of 1 each in the unit cell.

3.3 Respect neighbor constraints

At this point, we have successfully encoded the colors for each region via logical qubits, and mapped the logical qubits and couplers to physical unit cells of the D- Wave system. We now need to enforce the neighbor constraints. With some foresight we have chosen the unary encoding and logical to physical mapping so that this next step (and the following one, too) become very simple.

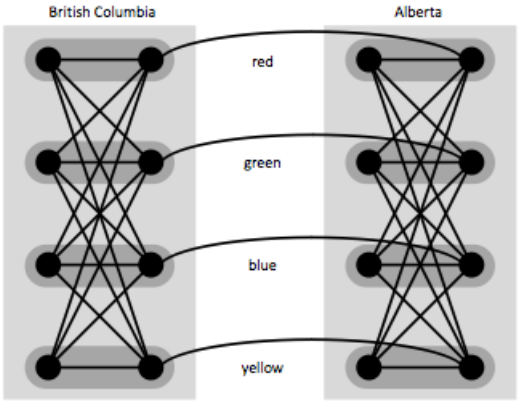

Figure 4: British Columbia and Alberta map to neighboring unit cells

Our job is to adjust the weights and strengths so that when, for example, the red qubit for British Columbia turns on and the red qubit for Alberta also turns on, the value of the objective function increases. The objective should stay constant when both these red qubits are turned off or also when one or the other of them (but not both) is turned on. Refer to the enumeration of two-qubit states in table 1. Assume that q_1 refers to the British Columbia's red qubit and q_2 refers to Alberta's red qubit. By setting $a_1 = 0$ and $a_2 = 0$ and $b_{12} = 1$ we lift the objective for the fourth state, which we need to penalize, and leave the other three states at an objective of 0.

To implement this penalty, we need to find a physical coupler that connects British Columbia's red qubit to Alberta's red qubit. This requires us to look at a slightly larger portion of the fabric of physical qubits and couplers comprising the D-Wave system. Figure 4 shows two neighboring unit cells in the D-Wave fabric and highlights the chains in each unit cell that represent the four logical qubits, one for each color. Most importantly, for the current step, this figure also represents the physical couplers connecting two adjacent unit cells as arcs. It is clear that these couplers are ideally positioned to implement the portion of

the objective that ensures that if the chain representing color *i* is turned on in one unit cell and the chain representing color *i* in the neighboring unit cell is also turned on, this state will be penalized. All we have to do is to set the strength associated with the four arc-shaped couplers to 1.

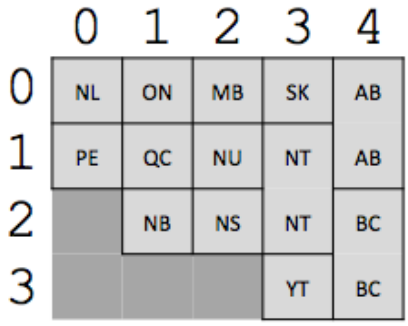

Figure 5: Mapping of Canada's 13 regions to a portion of the D-Wave unit cell array; row and column indices are 0-based; regions are labelled using standard two-letter postal codes

3.4 Clone regions as necessary

This last step in mapping the coloring problem to our programming model is necessitated by the neighbor relations in the map itself and the connectivity of unit cells in the D-Wave system. To appreciate this problem, note that British Columbia, Alberta and the Northwest Territories are all neighbors (see figure 1). Next, observe that the unit cells in the D-Wave system are configured in a two-dimensional checkerboard array. The configuration of these three regions cannot be mapped to unit cells while preserving the neighbor relation.

We could certainly assign British Columbia and Alberta to unit cells that neighbor each other horizontally (as in figure 4) and we could also assign the Northwest Territories to a unit cell positioned vertically above British Columbia. This configuration means that Alberta is not a direct neighbor of the Northwest Territory in our unit cell array and hence there are no physical couplers available to ensure that the same color qubits in these two regions will not be simultaneously activated.

Solve this last problem by introducing *clones*. These are analogous to the chains of physical qubits that represent a single logical qubit. In this case, use multiple unit cells to represent the color of a single region. Just as with chains of physical qubits, the motivation for cloning is that extending the footprint of a single region in the unit cell array provides more neighbors for a cloned region. This allows us to enforce neighbor constraints that arise from Canada's map that do not transfer directly to the unit cell array.

Figure 5 shows a mapping of Canada's 13 regions to unit cells with Alberta (AB), British Columbia (BC) and the Northwest Territories (NT) all cloned. It is easily checked with reference to figure 1 that each neighbor relation from the map of Canada corresponds to some adjacent pair of unit cells in the D-Wave cell array.

To complete this step, we must adjust the strengths for the intercell couplers so that the color assigned to Alberta in the unit cell in row 0 and column 4 matches the color assigned to Alberta in the unit cell in row 1 and column 4. Solve this problem in exactly the same way as for the chains of physical qubits representing a logical qubit. For the physical coupler connecting the red qubit

in Alberta's top cell to the red qubit in Alberta's bottom cell, adjust the strength of the coupler to -2 and add a weight of 1 to the two physical qubits connected via the coupler. Do the same thing for each of the other colors and proceed likewise for British Columbia and the Northwest Territories, which have also been cloned.

4 Conclusion

We have described the four steps required to transform the problem of generating a valid coloring for the regions of Canada into a single QMI for the D-Wave system. These four steps follow naturally from the decision to represent colorings via a unary encoding scheme, which requires 13*C* qubits to represent the possible *C*-colorings of the thirteen regions of Canada.

Implementation of these four steps in a conventional programming language (i.e.,C) is covered in the appendix. Regardless of language, one uses standard constructs to generate the weights and strengths of the QMI. Special library routines are then used to pass the QMI to the D-Wave system and retrieve samples from the resulting distribution.

Three conclusions can be drawn immediately from this exercise. First, the work of mapping this problem to a QMI could be simplified via routines which create embeddings of a logical formulation to a physical QMI. Second, the size of a map that can be colored using this strategy is limited by the number of unit cells available in the D-Wave system. A more scalable strategy would include the ability to divide larger maps into *chunks* which can be handled individually. Results from several chunks would be synthesized to create a coloring of a larger map. Finally, the D- Wave unit cell is large enough to handle four colors with this encoding, but this scheme must be modified to handle more general problems that may require more than four colors.

 | 3033 Beta Ave. Burnaby BC V5G 4M9 | Tel: +1 604-630-1428 | www.dwavesys.com

Appendix: Code fragments in C

This appendix contains portions of a C program that implements map coloring for Canada's 13 regions on a 512-qubit D-Wave system. Not shown are portions of the code necessary to connect to the system and configure parameters associated with executing the QMI. What *is* shown is most of the body of a function **setup_unit_cell(int row, int col)** which initializes the weights and strengths described earlier for each unit cell of the D-Wave system. Each of the 13 regions of Canada is mapped to one or two unit cells, so we must provide the row and column location of the unit cell to this function so that the correct neighbor and clone relations deriving from steps 3 and 4 are incorporated into the unit cell's weights and strengths.

Before the code below is invoked we allocate an array **weight** whose index is the qubit number, which ranges from 0 to 511 for the D-Wave system used for this example. Within each unit cell (see figure 3) the qubits are sequentially numbered, starting at the top of the left column qubits and continuing with the right column qubits. The qubit number for the upper left cell in the two dimensional array of unit cells making up the D-Wave system begins at 0. The immediate neighbor to the right of this unit cell begins with qubit number 8, and so on in row-major order.

The macro **DW_QUBIT** converts its four arguments into a qubit number. This allows us to convert from one system of numbering qubits - via rows and columns which identify the unit cell, a column specifier which is '**L**' for left and '**R**' for right and an offset within the column - into the single qubit number, which indexes into the **weight** array.

Step 1 ensures that of the *C* logical qubits representing the *C* colors of a region, exactly one turns on. Achieve this by setting the weight for each of the *C* qubits to –1 and the strengths connecting all pairs of logical qubits to 2, as discussed earlier. Step 2 forces us to associate a chain of two physical qubits for each of the *C* logical qubits created in step 1, so the weight of –1 is split equally across the two physical qubits.

In addition to splitting the weight across the two physical qubits that form the chain which represents a logical qubit, we also split the strength, 2, across two physical couplers. This is because there are two physical couplers connecting the British Columbia red chain to the British Columbia green chain (see figure 4). Splitting the strength means we increment the appropriate locations in the strength array with a value of 1.

In the same way that we allocate a **weight** array before executing the code block below, we also allocate a **strength** array, whose index is a physical coupler number.

All the couplers in the system can be classified as *intracell* or *intercell*. The macro **DW_INTRACELL_COUPLER** maps allows us to convert from one system of numbering couplers - via rows and columns which identify a unit cell and a pair of offsets for the left and right column qubits connected via the coupler - into a single index. Only those intracell couplers for which *i* and *j* are not equal link different chains and so we increment the associated strengths in this case only.

```
/* STEP 1: turn on one of C qubits */
/* Handle weights                  */
for (i=0; i<C; ++i)
  {
    weight[DW_QUBIT(row,col,'L',i)] += -0.5;
    weight[DW_QUBIT(row,col,'R',i)] += -0.5;
  }
/* Handle strengths */
for (i=0; i<C; ++i)
  for (j=0; j<C; ++j)
    if (i != j)
      strength[DW_INTRACELL_COUPLER(row,col,i,j)] += 1;
```

Step 2 creates chains consisting of two physical qubits for each logical qubit. Accomplish this via incrementing the **weight** for the two physical qubits by 1 and the **strength** for the connecting physical couplers by -2. Note that we are incrementing the same **weight** and **strength** arrays as in step 1, thus we must make sure to increment the relevant location in **weight** rather than to simply overwrite it. Both the **weight** and **strength** arrays must be initialized to zero before applying these increments.

```
/* STEP 2: create chains */
for (i=0; i<C; ++i)
  {
    weight[DW_QUBIT(row,col,'L',i)] += 1;
    weight[DW_QUBIT(row,col,'R',i)] += 1;
    strength[DW_INTRACELL_COUPLER(row,col,i,i)] += -2;
  }
```

Steps 1 and 2 did not involve the neighbor or clone relationships and so only used the row and column values to make sure that the correct qubit number and coupler number were calculated before incrementing the **weight** and **strength** arrays. In step 3 the neighbor relation is mapped to couplers that connect one unit cell to a neighboring unit cell. This is necessary if the regions mapped to two neighboring unit cells in the D-Wave array are also neighbors in the Canada map.

To check whether this is true, index into the **cell_region** array using the **row** and **col** inputs to the function. Do this for two unit cells that are horizontal neighbors in the D-Wave array to obtain the regions mapped to these two unit cells. Then, lookup up these two regions in the **is_neighbor** array, which captures the 15 total neighbor relations in the Canada map. If this array contains a true value, we must compute strength values and place them in the strength array.

```
/* STEP 3: enforce neighbor constraints */
/* Handle horizontal neighbors:         */
if (col < DW_N-1 &&
    is_neighbor(cell_region[row][col],cell_region[row][col+1
  for (i=0; i<C; ++i)
    strength[DW_INTERCELL_COUPLER(row,col,'H',i)] += 1;
```

The above code fragment handles horizontally neighboring unit cells and the below code fragment handles vertically neighboring unit cells. In the above code fragment first check to make sure that the col value does not refer to the right-most column in the D-Wave array so that we will not reference an invalid memory location in the **cell_region** array. Likewise, below check that **row** is not the bottom row of the array. The **DW_INTERCELL_COUPLER** macro computes the physical coupler number given the row and column of a unit cell, the direction (either 'H' for horizontal or 'V' for vertical) of the intercell coupler, and the offset within the unit cell at which the coupler is located.

```
/* Handle vertical neighbors: */
if (row < DW_M-1 &&
    is_neighbor(cell_region[row][col],cell_region[row+1][col]))
  for (i=0; i<C; ++i)
    strength[DW_INTERCELL_COUPLER(row,col,'V',i)] += 1;
```

The logic of step 4 is quite similar to that of step 3, except that we must check for a pair of neighboring unit cells which represent the same region. Thus, rather than use the **is_neighbor** array, we simply compare the regions represented by the two neighboring unit cells. If they match then increment elements in **weight** and **strength** to cause these two unit cells to have the same pattern of qubits "on" and "off". Weights are incremented for the two physical qubits on either end of the coupler by 1 and the strength of the physical coupler is incremented by -2.

```
/* STEP 4: create clones     */
/* Handle horizontal clones: */
if (col < DW_N-1 &&
    cell_region[row][col] == cell_region[row][col+1])
  for (i=0; i<C; ++i)
    {
      weight[DW_QUBIT(row,col  ,'R',i)] += 1;
      weight[DW_QUBIT(row,col+1,'R',i)] += 1;
      strength[DW_INTERCELL_COUPLER(row,col,'H',i)] += -2;
    }
/* Handle vertical clones:   */
if (row < DW_M-1 &&
    cell_region[row][col] == cell_region[row+1][col])
  for (i=0; i<C; ++i)
    {
      weight[DW_QUBIT(row  ,col,'L',i)] += 1;
      weight[DW_QUBIT(row+1,col,'L',i)] += 1;
      strength[DW_INTERCELL_COUPLER(row,col,'V',i)] += -2;
    }
```

After all four steps have completed, the **weight** and **strength** arrays are completely initialized. Straightforward library code (not shown here) transfers these arrays into the QMI and then passes the QMI to the D-Wave system for execution. In addition to the weights and strengths, there are a few other pieces of information that complete the QMI. The most important is the number of samples to retrieve from the distribution.

Table 3 shows three trials at each of $C = 2, 3, 4$. The QMI for each C value was executed three times, with a sample size of 1000. The qubit values in each sample were scanned to determine the number of unique samples (because a given pattern of qubit values can occur more than one time amongst the samples) and the number of valid colorings were computed.

C	valid colorings/unique samples
2	0/37, 0/45, 0/34
3	252/417, 228/371, 258/393
4	837/978, 812/947, 801/955

Two colors is not enough, but three (or four) colors suffice to create a valid coloring of Canada's 13 regions.