Embedding logical functions into the Chimera graph

Katalin Friedl[1]

Department of Computer Science and Information Theory
Budapest University of Technology and Economics
H-1111 Budapest, Műegyetem rakpart 3., Hungary
friedl@cs.bme.hu

László Kabódi[1]

Department of Computer Science and Information Theory
Budapest University of Technology and Economics
H-1111 Budapest, Műegyetem rakpart 3., Hungary
kabodil@cs.bme.com

Abstract: The Chimera graph is used in the D-Wave quantum annealer machines. Although there is a debate whether these machines are truly quantum, it is still meaningful to investigate the corresponding computational model. In this paper we show a method to embed some logical functions into the Chimera graph which can be used to solve the SAT problem using a quantum annealer.

Keywords: adiabatic quantum computation, Chimera graph, D-Wave Systems

1 Introduction

Quantum computing is a promising field of algorithmic research. The best known results are the algorithms of Grover and Shor. Grover's search finds a marked element in a list of N unordered elements in only $O(\sqrt{N})$ quantum steps and Shor's algorithm finds a prime factor of a composite number in expected polynomial time.

The model of quantum computation that is used most of the time, and also in these two famous algorithms, is the circuit model when the algorithm is built up from a small set of quantum gates, similarly to the classical model that is based on a basic set of Boolean gates.

Adiabatic computing [5] is a continuous model for quantum computations. It uses a physical process to perform quantum annealing. The problem to be solved is phrased as an optimization problem. The algorithm starts in an initial state H_i that should be an easily obtained ground state of the system. During the computation the starting Hamiltonian H_i evolves adiabatically, slowly changing but staying in ground state to reach H_p. The solution is encoded in H_p. In this type of computation the time depends on the physical process, and a challenge is to define (and create) the right Hamiltonians. It was shown in [1] that this adiabatic model is equivalent to the gate model.

Currently there is only one type of commercially available computer based on this idea, produced by D-Wave Systems, although there are doubts whether these machines really perform quantum computations. However, their structure provides an interesting computational model. In this model we do not have to deal with the Hamiltonians, the main task is to embed problems into special type of graphs (Chimera graphs).

Such embeddings are given for the general or this special adiabatic model in a few papers [2, 4, 6, 7]. They show how to represent for example an input graph in this model.

In paper [3] the first few NP-complete problems of Karp are embedded into the general adiabatic setting. Here the 3SAT problem is handled by reduction from the maximum independent set problem using the general adiabatic framework.

[1]Research is supported by OTKA-108947.

Our goal in this paper is to show a direct embedding of logical functions into the Chimera graph, specifying a possible setting for the weights of the graph.

Section 2 describes the architecture (Chimera graph), the parameters and the discrete optimization problem arising in the model of D-Wave machines. Section 3 describes the general methods used in our approach. Section 4 shows how to use these to compute the OR function of n bits, the next section describes the case of AND. Section 6 sketches how to put these together to obtain an embedding of any CNF formula.

2 The programming model of the D-Wave machine

The underlying optimization problem is the quadratic unconstrained binary optimization (QUBO) problem. For this a graph is given on N nodes, its edge set is denoted by E. The nodes and edges have weights α_i and $\beta_{i,j}$, respectively. In the corresponding QUBO problem there is a $\{0,1\}$ variable z_i to each node and the goal is to find the minimum of $\sum_{i=1}^{N} \alpha_i z_i + \sum_{\{i,j\}\in E} \beta_{i,j} z_i z_j$.

The hardware in the case of D-Wave machines uses variables $y_i \in \{-1,+1\}$, but it is easy to transform from z_i to y_i and vice versa. We will mostly use $\{0,1\}$ variables, but some ideas are easier to see in $\{-1,+1\}$.

In the case of D-Wave the underlying graph is not a complete graph. This makes formulation of problems in this setting more challenging. The computer uses a Chimera graph, which is an $m \times m$ grid of complete bipartite graphs $K_{n,n}$. (An existing choice is $m = 12, n = 4$.) Figure 1 shows a Chimera graph with a 3×3 grid and $K_{4,4}$ (a 3-4-Chimera graph). Programming the machine means setting the constants α_i and $\beta_{i,j}$. The hardware then finds the minimum of the QUBO and outputs an optimal choice for z_i.

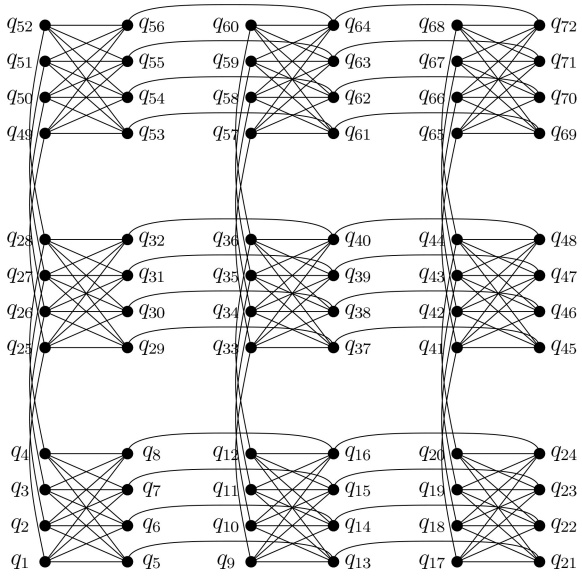

Figure 1: A 3-4-Chimera graph. Image from [8].

In a Chimera graph a node can be identified using 3 indices. The first two describe the position in the grid, the third gives its place in the corresponding bipartite graph. In the $K_{n,n}$ we number the nodes starting on the left side from top to bottom and continuing on the right side from top to bottom. For example $x_{2,3,5}$ is in the second row third column of the grid and the fifth node of the bipartite graph (that is q_{48} in Figure 1).

Using this notation the set of nodes is $\{x_{i,j,k} \mid 1 \leq i,j \leq m, 1 \leq k \leq 2n\}$. There are three kinds of edges in the graph. There are the edges of the complete bipartite graphs. The other two kinds are going between bipartite graphs. One type is the vertical connections, where $x_{i,j,k}$ is connected to $x_{i-1,j,k}$ and $x_{i+1,j,k}$ if $k \leq n$, and $2 \leq i \leq m-1$. The first and last one have only one vertical edge, $x_{1,j,k}$ is connected to $x_{2,j,k}$ and $x_{m,j,k}$ is to $x_{m-1,j,k}$. The other type is the horizontal connections, where $x_{i,j,k}$ is connected to $x_{i,j-1,k}$ and $x_{i,j+1,k}$ if $k > n$ and and $2 \leq j \leq m-1$. The first and last one have only one horizontal edge, $x_{i,1,k}$ is connected to $x_{i,2,k}$ and $x_{i,m,k}$ is to $x_{i,m-1,k}$. Notice that only the nodes on the left side of the $K_{n,n}$ have vertical edges and the nodes on the right side have horizontal ones.

In this area, embedding a graph G into a graph H means that for every vertex v of G there is a subset X_v of the vertices of H with the properties that for different vertices the sets X_v are disjoint, X_v induces a connected graph in H, and if there is an edge in G between v and w then there is an $a \in X_v$ and a $b \in X_w$ that a and b are connected be an edge in H.

One can embed a complete graph in this structure [6] which is useful to solve graph problems and also makes the definition of QUBO problems easier. But if the problem does not need a complete graph, there may be embeddings with less overhead or a cleaner design.

3 Overview of the method

The idea behind the method is to create a modular design, where one can take the appropriate modules and put them together to form arbitrary logical functions. Before we describe selected gates, we discuss the broad structure of our method.

Definition 1 *A* module *is a self-contained implementation of a small logical function.*

In our case each module is a $K_{n,n}$. Every node has a value $z_i \in \{0,1\}$.

Definition 2 *A* state *of a module is the value of its nodes.*

Definition 3 *The* value *of a state is* $\sum_{i=1}^{N} \alpha_i z_i + \sum_{\{i,j\} \in E} \beta_{i,j} z_i z_j$ *where* z_i *are the values of the nodes and* E *is the set of edges of the module.*

On the left side of the module there are three kinds of nodes: input nodes, one output node and some or none other nodes. The other nodes are not used in the module, they are only there because of the hardware. The input nodes correspond to the variables of the logical function and the output node to the value of the function.

The value of the output node is called the output of the module. The goal is to set the weights of the module such that the value of the module is minimal if and only if the output is equal to the value of the logical function.

Definition 4 *A state is* valid *if the output is the value of the function and the value of any node* $x_{i,j,k}$ *on the right side is equal to the negated value of* $x_{i,j,k-n}$ *on the left side. Otherwise the state is* invalid.

Definition 5 *We call a state* true *if it is valid and the value of the function is true. We call it* false *if it is valid and the value of the function is false.*

For an example let us examine the $\begin{pmatrix} z_1 & z_4 \\ z_2 & z_5 \\ z_3 & z_6 \end{pmatrix}$ state of a $K_{3,3}$ OR module, where z_1 and z_2 are the inputs and z_3 is the output. The state $\begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{pmatrix}$ is true, $\begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$ is false and $\begin{pmatrix} 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}$ is invalid.

In our construction the weights of an edge or node depends only on its type. Let us assume that on the left side the last node is the output node, the others input nodes. We use the following parameters:

- a: the weight of all edges between $x_{i,j,k}$ and $x_{i,j,k+n}$ where $k \leq n$
- b: the weight of all edges between $x_{i,j,k}$ and $x_{i,j,\ell}$ where $k \neq \ell$, $k < n$, $\ell > n$
- c: the weight of all edges between $x_{i,j,k}$ and $x_{i,j,2n}$, where $k < n$
- d: the weight of all edges between $x_{i,j,k}$ and $x_{i,j,n}$, where $k > n$ and $k \neq 2n$
- L: the weight of all nodes $x_{i,j,k}$, where $k \leq n$
- R: the weight of all nodes $x_{i,j,k}$, where $k > n$

More precisely we work with modules that are $K_{3,3}$, where node number 1 and 2 are input nodes and 3 is the output node. For the case of larger $K_{n,n}$ the construction can be easily transformed by setting all the weights not included in the $K_{3,3}$ to be 0.

The optimization problem searches the minimum state of the QUBO problem. Our goal is to set the value of previous parameters such that all true states have the same W_t value, all false states have the same W_f value and the value of any invalid state is at least W_i. Also we want $W_t = W_f - 1$ and $W_t \leq W_i - k$, to have a $k \geq 2$ gap between the true states and the invalid states.

There might be technical constraints on the values of the parameters, but we will disregard them.

4 Embedding a logical OR

First, let us describe a module for an OR of two logical variables. It in not difficult to check that the following parameters satisfy the constraints using $k = 2$: $a = 10, b = -2, c = 6, d = \frac{2}{3}, L = -3, R = -\frac{8}{3}$. Using these parameters with a $K_{3,3}$ the value of the true states are -9, the false state is -8 and the invalid states are at least -7.

Attaching two modules together is a simple additive step. To obtain an OR function with 3 variables we use two neighbouring modules of the grid. The first represents $r_1 = p_1 \vee p_2$ and the second $r = r_1 \vee p_3$.

In order to do this one has to be able to copy the value of a node to another node. Because the QUBO is a simple sum of the different products, one can simply set the value of a few external edges, without modifying the inside of a module.

To copy the result to a new node, we simply use a sufficiently large positive or negative edge, and compensate its effect on the connected nodes. It is easier to see how this works with variables $y_i \in \{-1, +1\}$. In this case to force two connected nodes to be the same, we must use a negative edge, otherwise a positive one. For the $\{0, 1\}$ case, we first transform the variables to $\{-1, +1\}$, then use the appropriate edge and we get the weights needed to the $\{0, 1\}$ case. By this method the edge weight $w_{i,j}$ of the $\{-1, +1\}$ case transforms to the case $z_i, z_j \in \{0, 1\}$ as follows: $(2z_i-1)(2z_j-1)w_{i,j} = 4z_iz_jw_{i,j}-2(z_i+z_j)w_{i,j}+w_{i,j}$. The last term does not depend on the variables z_i, so it is not important from the point of view of minimalization. The others mean the we need to add $-\frac{1}{2}$ times the weight of the edge to the weight of nodes it connects. As before, $w_{i,j}$ is negative when copying and negative for negation.

Because we set the value of the false state to be one more than the value of the true states, we must compensate for it, so we add one to the weight of the edge that copies the result of the first module to the second one. This ensures that the minimum states include the ones, where there are some false modules, but the overall value of the function is true.

5 The logical AND

The logical AND function can be obtained from the logical OR and negations. But we think the design is cleaner if we make a separate AND module.

Applying the same constraints to the logical AND function, we obtain the following parameters: $a = 10, b = -\frac{2}{3}, c = 1, d = 5, L = -3, R = -\frac{8}{3}$. We deliberately chose a, L and R the same as in the case

of OR. Using these parameters with a $K_{3,3}$ the value of the true states are -9, the false state is -8 as before, and the invalid states are at least -6.6667.

Attaching the modules together is almost the same as in the OR case. The main difference is that in the AND function the result is only true if all the variables are true, so we do not need to add one to the weight of the edge that copies the result. (We can, the results will be the same, but we do not need to.)

6 The SAT problem

Embedding a general SAT problem using the previous modules is easy if the grid is large enough. The logical function has to be in CNF form. Then each clause gets its own column in the grid.

Each variable of the formula has its own row. For a logical variable p_i all $x_{i,j,1}$ correspond to p_i and $x_{i,j,n+1}$ to $\neg p_i$. If the clause in the jth column needs the negated version of the variable, then instead of copying the value (by negative edge weight) to that column we use positive edge weight to obtain the negation of the variable.

To implement the whole CNF formula there are three kinds of modules. The OR module, the AND module, and a copy module. The copy module keeps the value of one of its inputs, and copies the other to its output.

During this process, because the kth node of one bipartite graph is only connected to the kth node of the neighbouring bipartite graphs, the input and the output must switch places alternately. With the help of this, in a column we can move the partial results to the literals included in that clause where the OR module can be applied.

The results of the OR modules are copied into one row, in which we use AND modules. One of the input nodes of these AND modules, coming from the OR modules are always at the same position. The other input node that corresponds to the the result of the previous AND and the output node switch places alternately as we move from one AND module to the next.

7 Conclusion

In this paper we proposed a framework for constructing modules from small logical functions and applied it to construct OR and AND modules. From these, we made an embedding for any CNF into the Chimera graph. For a CNF containing n variables and m clauses, we need a Chimera graph with $n+1$ rows and m columns, so a $\max(n+1, m)$-3-Chimera graph. This embedding is not optimal, the number of nodes can be reduced, but our goal was not to find the optimal embedding, rather a clean and simple one. Later research should be done to optimize these results.

References

[1] D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. LLoyd, O. Regev, Adiabatic quantum computation is equivalent to standard quantum computation, *SIAM J. Computing* **Vol. 37, pp 166-194** (2007)

[2] V. Choi, Minor embedding in adiabatic quantum copmutation: 1. The parameter setting problem *Quantum Information Processing* **Vol.7, pp. 193-209** (2008)

[3] V. Choi, Adiabatic quantum algorithms for the NP-complete maximum weight independent set, exact cover and 3SAT problems *arXiv:quant-ph/1004.2226* (2010)

[4] C.S. Claude, E. Claude, M.J. Dinneen, Guest column: Adiabatic quantum computing challenges *ACM SIGACT News* **Vol.45, pp 40-61** (2015)

[5] E. Fahri, J. Goldstone, S. Gutmann, M. Sipser, Quantum computation by adiabatic evolution, *arXiv:quant-ph/0001106* (2000)

[6] C. Klymko, B. D. Sullivan, T. S. Humble, Adiabatic quantum programming: Minor embedding with hard faults *Quantum Information Processing* **Vol.13, pp 709-729** (2014)

[7] E. G. Rieffel, D. Venturelli, B. OGorman, M. B. Do, E. M. Prystay, V. N. Smelyanskiy, Vadim N, A case study in programming a quantum annealer for hard operational planning problems *Quantum Information Processing* **Vol.14, pp 1-36** (2015)

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