Chapter 17
Solving the Broadcast Time Problem Using a D-wave Quantum Computer

Cristian S. Calude and Michael J. Dinneen

Abstract We illustrate how the D-Wave Two quantum computer is programmed and works by solving the Broadcast Time Problem. We start from a concise integer program formulation of the problem and apply some simple transformations to arrive at the QUBO form which can be run on the D-Wave quantum computer. Finally, we explore the feasibility of this method on several well-known graphs.

17.1 Introduction

D-Wave machines are "the world's first commercially available quantum computers." D-Wave Two is an adiabatic machine which operates on 512 qubits. Various teams based in academia and major companies like Lockheed Martin, NASA, Google, are exploring the computational capability of this machine as well as its potential applications.

Programming D-Wave is radically different from programming classical machines, like one's traditional home computer. In this paper we use the Broadcast Time Problem, an NP-complete problem, to illustrate how to develop programs for the D-Wave Two machine and how it operates. We start from a concise integer program formulation of our optimization problem and, through several intermediate phases, we convert it to the QUBO form, the formulation which can be run on the D-Wave computer. Finally we explore the feasibility of this method on several well-known graphs.

C.S. Calude (✉) · M.J. Dinneen
Department of Computer Science, University of Auckland, Auckland, New Zealand
e-mail: cristian@cs.auckland.ac.nz

M.J. Dinneen
e-mail: mjd@cs.auckland.ac.nz

A. Adamatzky (ed.), *Advances in Unconventional Computing*,
Emergence, Complexity and Computation 22,
DOI 10.1007/978-3-319-33924-5_17

17.2 Adiabatic Computing

The adiabatic model of quantum computing uses the propensity of physical systems—classical or quantum—to minimize their free energy. Quantum annealing is free energy minimization in a quantum system.

Its mathematical precursors are the Monte Carlo methods [1]—in which a problem is solved by generating repeated random samplings from a probability distribution, performing simple deterministic computations and aggregating the results—and the Metropolis-Hastings algorithm—a Markov chain Monte Carlo method useful when direct sampling is difficult [2]. In 1983 an analogy between minimizing the cost function of a combinatorial optimization problem—solved efficiently with the Metropolis-Hastings algorithm—and the slow cooling of a solid until it reaches its low energy ground state was discovered in [3]. The proposed method—called *simulated annealing* [3, 4]—is very simple: substitute the cost for energy and run the Metropolis-Hastings algorithm in a sequence of slowly decreasing temperature values, which makes the system progress through various energy states till, hopefully, it finds a global optimal answer.

An *adiabatic quantum computation* (AQC) is an algorithm that computes an exact or approximate solution of an optimization problem encoded in the ground state—its lowest-energy state—of a Hamiltonian (the operator corresponding to the total energy of the system). The algorithm starts at an initial state H_I that is easily obtained, then evolves adiabatically, i.e. by slowly changing to the Hamiltonian H_P. An example of evolution is $H = (1 - t)H_I + tH_P$ as the time t increases monotonically from 0 to 1. During the entire computation, the system must stay in a valid ground state. If the system can reach its ground state we get an exact solution; if it can only reach a local minimum, then we get an approximate solution. The slower the evolution process the better the approximate (possibly exact) solution is obtained.

The adiabatic quantum computing model shares the same paradigm as simulated annealing. The main difference is that simulated annealing is based on "thermodynamic energy" and quantum annealing is based on "quantum fluxuations" during a cooling process. One suggested advantage of quantum annealing is the ability to "quantum tunnel" out of some local optimal states,[1] as illustrated in Fig. 17.1.

AQC is based on the Born–Fock adiabatic theorem [5] which accounts for the adiabatic evolution of quantum states when the change in the time-dependent Hamiltonian is sufficiently slow [6]:

> *A physical system remains in its instantaneous eigenstate if a given perturbation is acting on it slowly enough and if there is a gap between the eigenvalue and the rest of the Hamiltonian's spectrum.*

The quantum adiabatic computation model and the gate quantum computation model—probably the most studied model of quantum computing—are polynomially time equivalent [7].

[1] D-Wave Two is capable of using it.

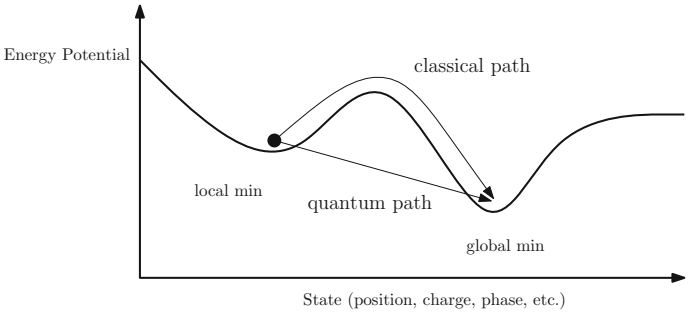

Fig. 17.1 Annealing with quantum tunneling

17.3 D-Wave Computers

The D-Wave computers are produced by the Canadian company D-Wave Systems: D-Wave One (2011) operates on a 128-qubit chipset; D-Wave Two (2013) works with 512 qubits [8]. D-Wave computers use quantum annealing to improve convergence of the system's energy towards the ground state energy of a *Quadratic Unconstrained Binary Optimization* (QUBO) problem. QUBO is an NP-hard mathematical problem consisting in the minimization of a quadratic objective function $f(\mathbf{x}) = \mathbf{x}^T Q\mathbf{x}$, where $\mathbf{x}$ is a n-vector of binary variables and Q is a symmetric $n \times n$ matrix:

$$x^* = \min_{\mathbf{x}} \sum_{i \geq j} x_i Q_{(i,j)} x_j, \text{ where } x_i \in \{0, 1\}.$$

The computer architecture consists of qubits arranged with a host configuration as a subgraph of a *Chimera graph*. A Chimera graph consists of an $M \times N$ two-dimensional lattice of blocks, with each block consisting of $2L$ vertices (a complete bipartite graph $K_{L,L}$), in total $2MNL$ variables. The D-Wave One has $M = N = L = 4$ for a maximum of 128 qubits. D-Wave qubits are loops of superconducting wire, the coupling between qubits is magnetic wiring and the machine itself is supercooled.

To index a qubit we use four numbers (i, j, u, k), where (i, j) indexes the (row, column) of the block, $u \in \{0, 1\}$ is the left/right bipartite half of $K_{L,L}$ and $0 < k < L$ is the offset within the bipartite half. Qubits indexed by (i, j, u, k) and (i', j', u', k') are neighbors if and only if

1. $i = i'$ and $j = j'$ and $[(u, u') = (0, 1)$ or $(u, u') = (1, 0)]$ or
2. $i = i' \pm 1$ and $j = j'$ and $u = u'$ and $u = 0$ and $k = k'$ or
3. $i = i'$ and $j = j' \pm 1$ and $u = u'$ and $u = 1$ and $k = k'$.

Figure 17.2 shows for $L = N = 4$ (and $M > 2$) the structure of an initial part of a Chimera graph where the two half partitions of the bipartite graphs $K_{L,L}$ (blocks) are drawn horizontally and vertically, respectively. The linear index (qubit id of the vertices) from the four tuple (i, j, u, k) is the value $2NLi + 2Lj + Lu + k$.

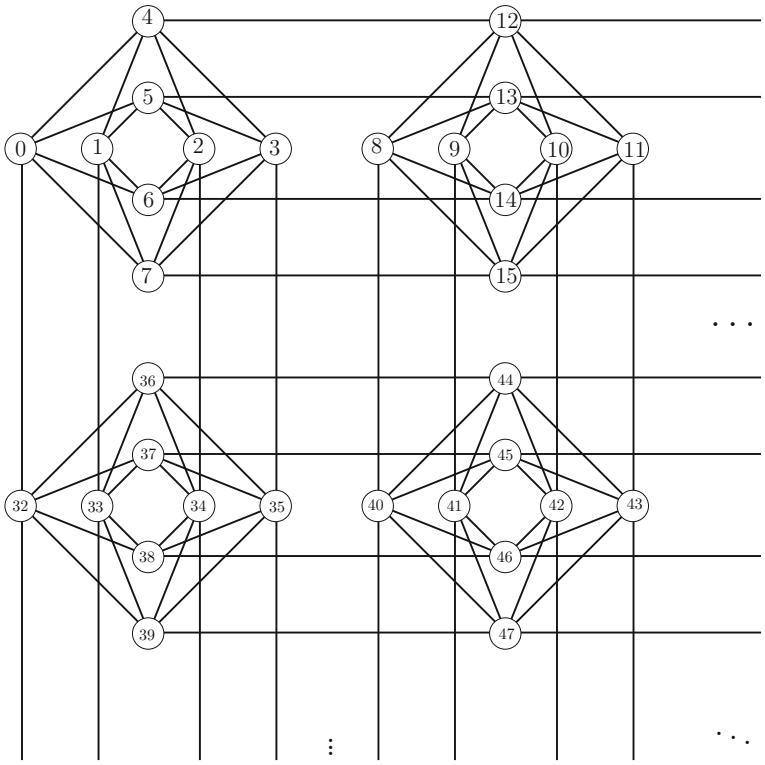

Fig. 17.2 D-Wave architecture: a subgraph of a Chimera graph with $L = N = 4$

17.4 The Broadcast Time Problem

Broadcasting concerns the dissemination of a message originating at one node of a network to all other nodes [9, 10]. This task is accomplished by placing a series of calls over the communication lines of the network between neighboring nodes. Each call requires a unit of time, a call can involve only two nodes and a node can participate in only one call per time step.

A *broadcast tree* for a vertex v (called the originator) of an undirected graph $G = (V, E)$ is an implicit rooted tree based on a sequence $V_0 = \{v\}, E_1, V_1, E_2, \ldots, E_t, V_t = V$ (*of broadcast height* t) such that each $V_i \subseteq V$, each E_i is an oriented subset of E, and for every $1 \leq i \leq t$: (1) each arc (u, w) in E_i has only the source u endpoint in V_{i-1}, (2) no two arcs in E_i share a common endpoint, and (3) $V_i = V_{i-1} \cup \{w \mid (u, w) \in E_i\}$.

The *Broadcast Time Problem* is the following: *Given a connected graph* $G = (V, E)$, *originator* $v \in V$ *and integer* t, *is there a broadcast tree* T_v *rooted at* v *with the height of* T_v *at most* t? This is a well-known NP-complete problem (see [ND49] of [11]), even for graphs of maximum vertex degree 3 (see [12]). The optimization

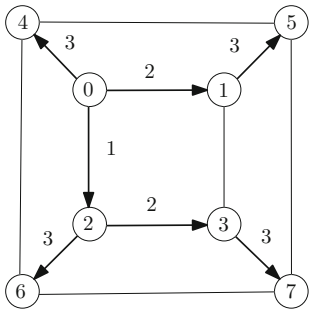

Fig. 17.3 The graph Q_3 with broadcast time 3

version of this problem is approximable within $O(\log^2 |V|/\log\log |V|)$, but is not expected to have a polynomial-time approximation scheme [13].

In the example shown in Fig. 17.3 (hypercube Q_3) an optimal broadcast tree is illustrated for the originator vertex 0.

The development of a quantum solution will be presented in a sequence of four phases, which are described in Sects. 17.5–17.8. We summarize this general solution approach for Q_3 in Sect. 17.9 before giving a detailed implementation of our broadcast time solution for K_2 in Sect. 17.10. Finally, we present some general experimental results for several other small common graphs (Sect. 17.11) and concluding comments (Sect. 17.12).

17.5 Integer Programming Formulation

In the first phase we present a simple formulation (i.e. polynomial-time reduction) of the Broadcast Time Problem with the originator fixed[2] at $v = 0$ as an *Integer Programming (IP) Optimization Problem* (see [14]). The input is a connected graph $G = (V = \{0, 1, \ldots, n-1\}, E)$ representing a network with $n = |V|$ vertices and $m = |E|$ edges. For the graph G, we use the following $n + 2m + 1$ variables:

- t is the required time to complete a broadcast,
- v_i is the time in $\{0, 1, \ldots, t\}$ in which the vertex $i \in V$ receives the message, $0 \leq i < n$,
- $b_{i,j}$ is a binary variable which is 1 if and only if the vertex i broadcasts to the vertex j (for each $\{i, j\} \in E$).

The objective function for our optimization problem is $\min(t)$, or equivalently, $\max(n - t)$.

Several families of constraints on the variables are now presented. First, the time t must be at most $n - 1$:

[2] Solving the problem for other originators can be easily done by relabelling the vertices of the graph or doing obvious modifications in the formulation below.

$$0 \leq t \leq n-1. \tag{17.1}$$

Every vertex receives the message at a time step at most t:

$$0 \leq v_i \leq t, \text{ for all } i \in V. \tag{17.2}$$

The originator vertex has no parent and every other vertex must have exactly one parent in the broadcast tree:

$$\sum_{j \neq 0} b_{j,0} = 0, \tag{17.3}$$

$$\sum_{j \neq i} b_{j,i} = 1, \text{ for all } i \in V \setminus \{0\}. \tag{17.4}$$

There are no broadcast cycles, that is for a child vertex, the informed time of the parent must be strictly less than its message received time:

$$b_{i,j}(1 + v_i - v_j) \leq 0, \text{ for all } \{i, j\} \in E. \tag{17.5}$$

Finally, every two child vertices $\{j, k\}$ informed by the same parent i must occur at different times:

$$b_{i,j} + b_{i,k} - (v_j - v_k)^2 \leq 1, \text{ for all } \{i, j\} \in E, \{i, k\} \in E \text{ with } j \neq k. \tag{17.6}$$

17.6 Binary Integer Programming Formulation

Next we convert all non-binary variables (in IP formulation) into binary variables. The following simple procedure converts an integer constrained variable $0 \leq x \leq D$ into a set of $O(\log D)$ binary variables $x_0, x_1, \ldots, x_c$ representing its binary representation:

$$x = x_0 + 2x_1 + 4x_2 + \cdots + 2^c x_c = \sum_{i=0}^{c} 2^i x_i,$$

where $x_i \in \{0, 1\}$ and $2^c \leq D < 2^{c+1}$. Each constraint of the form $x \leq D$ is replaced by the following equivalent constraint:

$$\sum_{i=0}^{c} 2^i x_i \leq D. \tag{17.7}$$

17.7 Linear Binary Integer Programming Formulation

Using standard techniques (e.g., see [15]) we convert the above quadratic binary IP formulation into a linear formulation. Each occurrence of a product of two binary variables xy is replaced by a new variable z_{xy} and the following two linear constraints:

$$0 \leq x + y - z_{xy} \leq 1, \tag{17.8}$$

$$-1 \leq 2z_{xy} - (x + y) \leq 0, \tag{17.9}$$

enforce $z_{xy} = xy$.

We can reduce the number of "product" binary variables by observing that for Eq. (17.5) we do not need to consider $j = 0$ and for Eq. (17.6) we can only consider vertices $j > 0$ and $k > 0$ with a common neighbor.

Note that the above reduction was automated and the Sage Mixed Integer Program Solver [16] was used to verify correctness of many small graphs [17].

17.8 QUBO Formulation

The first step to converting the current binary linear IP formulation to QUBO is to use a "standard form," where all inequalities are replaced with equalities by introducing slack variables [14].

The next step is to build an equivalent QUBO of the IP formulation and add rules to force all linear equation constraints to be satisfied when assigning 0/1 to the binary variables. Consider a linear equality constraint C_k of the form $\sum_{i=1}^{n} c_{(k,i)} x_i = d_k$ for $x_i \in \{0, 1\}$ with fixed integer constants $c_{(k,i)}$ and d_k. This equation is satisfied if and only if $\sum_{i=1}^{n} c_{(k,i)} x_i - d_k = 0$, or equivalently, if $\langle \mathbf{c}_k, \mathbf{x} \rangle - d_k = 0$, where $\mathbf{c}_k = (c_{(k,i)}, c_{(k,2)}, \ldots, c_{(k,n)})$ and $\langle \mathbf{c}_k, \mathbf{x} \rangle$ is the product of the vectors $\mathbf{c}_k$ and $\mathbf{x}$. If $\langle \mathbf{c}_k, \mathbf{x} \rangle - d_k$ is not zero we need to have a penalty greater than the maximum feasible value of t, which is n. Thus, we can construct the following QUBO that is equivalent to the IP formulation of the Broadcast Time Problem:

$$x^* = \min_{\mathbf{x}} \left(t + n \cdot \sum_{k} \left(\langle \mathbf{c}_k, \mathbf{x} \rangle - d_k \right)^2 \right), \text{ where } x_i \in \{0, 1\}.$$

Note first that the variable t is obtained from the variables used in Eq. (17.1) of Sect. 17.5 and is added to the other QUBO entries in Q from the set of linear constraints C_k. The QUBO constants for the binary variables representing t will be powers of 2, as given by Eq. (17.7). Second, any term d_k^2 in the square terms of $(\langle \mathbf{c}_k, \mathbf{x} \rangle - d_k)^2$ which does not involve a variable x_i can be ignored since those additive terms are independent of any assignment of variables (i.e. we have a fixed additive QUBO *offset* to the objective solution). Third, since variables x_i are binary,

we have $x_i^2 = x_i$ and the constants for those terms are included in the main diagonal entries of Q. Finally, the conversion from an arbitrary Broadcast Time Problem instance to QUBO was automated [17].

17.9 Q_3 Example

In this section we illustrate the quantum solution phases for Q_3. In the first phase (IP formulation) we get 33 integer variables (the variable t, eight variables of the type v_i, and 24 variables of the type $b_{i,j}$) and 65 quadratic constraints. These constraints are shown in the appendix of this chapter.

The conversion to the binary formulation results in 51 ($= 33 + 2 \cdot 9$) binary variables as we need three binary variables for each of the previous integer variables t, $v_0, \ldots, v_7$. The number of constraints stays the same but each gets expanded with more variables. For example, $x_1 \leq x_0$ becomes $-x'_0 + x'_3 - 2x'_1 + 2x'_4 - 4x'_2 + 4x'_5 \leq 0$.

The next conversion (Sect. 17.7) produces 447 binary variables and 851 linear constraints. Finally, the conversion to QUBO generated 999 slack variables, so in total 1446 binary variables: they represent the number of logical qubits for our QUBO formulation. Full details for our phases 2 through 4 may be found in [17].

To be able to solve this QUBO problem on D-Wave we need one more step to encode the theoretical problem on physical hardware (see [18]) which will be illustrated in the next section with a feasible example for the D-Wave Two.

17.10 K_2 Example

We present both the final IP formulation (see Table 17.1) and QUBO matrix Q (see Table 17.2) for the Broadcast Time Problem for the graph K_2 of one edge. The total number of binary variables is 22 (13 of them are slack variables) and the QUBO offset is 12. When run on the D-Wave simulator [18] (without embedding onto the hardware, which has limited qubit connections) we get this expected result:

```
answer={ 'energies': [-11.0],
         'solutions': [[1,0,0,1,1,0,0,1,0,0,1,1,0,0,0,
                        0,1,0,1,0,0,0]] }
```

When we add the offset 12 to the minimum energy state we get our expected broadcast time of 1. We can also see that $t = x_1 = 1$, $b_{0,1} = x_7 = 1$ and $b_{1,0} = x_8 = 0$, which indicates a valid broadcast tree from the obtained solution x^*.

To actually run this QUBO instance on the D-Wave machine we need to find a minor-containment embedding on the actual physical qubit hardware (the Chimera graph). One valid heuristic is to map each logical qubit to a path of physical qubits.

Table 17.1 Final Binary Integer Program for broadcasting in K_2

Integer program constraints	Comments
$x_0 + x_1 = 1$	x_0 is objective variable t and x_1 is a slack variable
$-x_0 + x_2 + x_3 = 0$	x_2 is vertex variable v_0; Eq. (4)
$-x_0 + x_4 + x_5 = 0$	x_4 is vertex variable v_1; Eq. (4)
$x_6 = 0$	x_6 is broadcast variable $b_{1,0}$; Eq. (5)
$x_7 = 1$	x_7 is broadcast variable $b_{0,1}$; Eq. (6)
$x_2 + x_7 - x_8 + x_9 = 1$	x_8 is for product $b_{0,1}v_0$ with Eq. (10)
$-x_2 - x_7 + 2x_8 + x_{10} = 0$	Equation (7) with (11)
$x_4 + x_7 - x_{11} + x_{12} = 1$	x_{11} is for product $b_{0,1}v_1$
$-x_4 - x_7 + 2x_{11} + x_{13} = 0$	Equation (7)
$x_4 + x_6 - x_{14} + x_{15} = 1$	x_{14} is for product $b_{1,0}v_1$
$-x_4 - x_6 + 2x_{14} + x_{16} = 0$	Equation (7)
$x_2 + x_6 - x_{17} + x_{18} = 1$	x_{17} is for product $b_{1,0}v_0$
$-x_2 - x_6 + 2x_{17} + x_{19} = 0$	Equation (7)
$x_7 + x_8 - x_{11} + x_{20} = 0$	Equation (8)
$x_6 + x_{14} - x_{17} + x_{21} = 0$	Equation (8)

Table 17.2 Final QUBO matrix Q for broadcasting in K_2

3	4	−4	−4	−4	−4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	**−2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
−4	0	**2**	4	0	0	8	8	−12	4	−4	0	0	0	0	0	0	−12	4	−4	0	0
−4	0	4	**2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
−4	0	0	0	**2**	4	8	8	0	0	0	−12	4	−4	−12	4	−4	0	0	0	0	0
−4	0	0	0	4	**2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	8	0	8	0	**4**	0	0	0	0	0	0	0	−8	4	−4	−16	4	−4	0	4
0	0	8	0	8	0	0	**0**	−8	4	−4	−16	4	−4	0	0	0	0	0	0	4	0
0	0	−12	0	0	0	0	−8	**16**	−4	8	−4	0	0	0	0	0	0	0	0	4	0
0	0	4	0	0	0	0	4	−4	**−2**	0	0	0	0	0	0	0	0	0	0	0	0
0	0	−4	0	0	0	0	−4	8	0	**2**	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	−12	0	0	−16	−4	0	0	**16**	−4	8	0	0	0	0	0	0	−4	0
0	0	0	0	4	0	0	4	0	0	0	−4	**−2**	0	0	0	0	0	0	0	0	0
0	0	0	0	−4	0	0	−4	0	0	0	8	0	**2**	0	0	0	0	0	0	0	0
0	0	0	0	−12	0	−8	0	0	0	0	0	0	0	**16**	−4	8	−4	0	0	0	4
0	0	0	0	4	0	4	0	0	0	0	0	0	0	−4	**−2**	0	0	0	0	0	0
0	0	0	0	−4	0	−4	0	0	0	0	0	0	0	8	0	**2**	0	0	0	0	0
0	0	−12	0	0	0	−16	0	0	0	0	0	0	0	−4	0	0	**16**	−4	8	0	−4
0	0	4	0	0	0	4	0	0	0	0	0	0	0	0	0	0	−4	**−2**	0	0	0
0	0	−4	0	0	0	−4	0	0	0	0	0	0	0	0	0	0	8	0	**2**	0	0
0	0	0	0	0	0	0	4	4	0	0	−4	0	0	0	0	0	0	0	0	**2**	0
0	0	0	0	0	0	4	0	0	0	0	0	0	0	4	0	0	−4	0	0	0	**2**

One such example is given below where our 22 logical qubits, labeled 0 to 21, become 50 active hardware qubits on D-Wave Two's Chimera graph with $L = 4$, $N = M = 8$.

```
'embedding=': [ 0=[224, 226, 228], 1=[230], 2=[276,
283, 284, 288, 292],  3=[290], 4=[227, 291, 348, 355,
356, 357], 5=[229], 6=[336, 338, 341, 347, 349], 7=
[293, 297, 301, 361], 8=[294, 296, 302], 9=[289], 10=
[300], 11=[298, 362, 365], 12=[359, 367], 13=[364],
14=[345, 351],15=[344], 16=[346], 17=[275, 277, 281,
285, 339], 18=[272], 19=[274], 20=[303], 21=[343] ]
```

This best energy solution of -11 is also obtained when we run it on an actual D-Wave Two machine. This optimal answer occurs about 33 % of the time for our trials of about 1000 runs. In the other cases, the machine did not converge to the optimal ground-state energy.

17.11 Experimental Results

We have produced QUBO representations of the Broadcast Time Problem for several small common graphs using the above IP formulation procedure. Tables 17.3 and 17.4 summarize them for some small common graph families and known special graphs (all graphs can be obtained from Sage [16, 17]). Recall that for non-symmetric graphs we initiate the broadcast at vertex labeled 0, using the vertex labels given by Sage's adjacency lists. In these tables, columns 2 and 3 (Integer Variables and Quadratic Constraints) indicate the size of the IP formulation presented in Sect. 17.5. Next, columns 4 and 5 (Binary Variables and Binary Constraints) indicate the size of the IP formulation given in Sect. 17.7. Finally, columns 6–8 (Slack Variables, Logical Qubits and Chimera/Physical Qubits) indicate the size of the final QUBO representation described in Sect. 17.8. Using this approach, the number of logical qubits equals the number of binary variables plus the number of slack variables.

17.12 Conclusions

In this paper we have shown the process of converting a well-known combinatorial optimization problem, the Broadcast Time Problem, to a QUBO form that can be solved on an adiabatic quantum computer like the D-Wave Two. Our procedure of using an integer programming formulation (e.g., standard polynomial-time reduction) can be easily applied to other hard problems. However, this straightforward approach does require a large number of qubits for relatively small input graph instances. Future work is required to reduce this overhead. One area of study is

Table 17.3 Number of qubits required for some small graphs families

Graph	Order	Size	Integer variables	Quadratic constraints	Binary variables	Binary constraints	Slack variables	Logical qubits	Chimera qubits
C3	3	3	10	16	50	86	96	146	394
C4	4	4	13	21	74	131	146	220	662
C5	5	5	16	26	178	324	366	544	3258
C6	6	6	19	31	240	443	495	735	4164
C7	7	7	22	36	311	580	642	953	
C8	8	8	25	41	391	735	807	1198	
C9	9	9	28	46	778	1484	1608	2386	
C10	10	10	31	51	944	1809	1948	2892	
C11	11	11	34	56	1126	2166	2320	3446	
C12	12	12	37	61	1324	2555	2724	4048	
Grid2 × 3	6	7	21	37	254	472	543	797	4306
Grid3 × 3	9	12	34	65	832	1597	1816	2648	
Grid3 × 4	12	17	47	93	1414	2745	3084	4498	
Grid4 × 4	16	24	65	133	2420	4737	5252	7672	
Grid4 × 5	20	31	83	173	5537	10909	11815	17352	
K2	2	1	5	7	9	15	13	22	47
K3	3	3	10	16	50	86	96	146	394
K4	4	6	17	33	94	171	202	296	1378
K5	5	10	26	61	248	469	606	854	7973
K6	6	15	37	103	366	713	981	1347	
K7	7	21	50	162	507	1014	1482	1989	
K8	8	28	65	241	671	1375	2127	2798	
K9	9	36	82	343	1264	2591	4200	5464	
K10	10	45	101	471	1574	3279	5588	7162	
K2 × 1=P2	3	2	8	12	36	59	64	100	170
K1 × 2=S2	3	2	8	12	40	68	76	116	238
K2 × 2=C4	4	4	13	21	74	131	146	220	662
K2 × 3	5	6	18	32	192	353	414	606	4823
K3 × 3	6	9	25	49	282	529	633	915	
K3 × 4	7	12	32	69	381	727	894	1275	
K4 × 4	8	16	41	97	503	973	1227	1730	
K4 × 5	9	20	50	129	976	1906	2432	3408	
K5 × 5	10	25	61	171	1214	2391	3124	4338	
K5 × 6	11	30	72	118	1468	2914	3896	5364	
K6 × 6	12	36	85	277	1756	3511	4804	6560	
S2=K1 × 2	3	2	8	12	40	68	76	116	238
S3	4	3	11	18	64	114	130	194	505
S4	5	4	14	25	164	301	354	518	3711
S5	6	5	17	33	226	423	501	727	5120
S6	7	6	20	42	297	564	672	969	
S7	8	7	23	52	377	724	867	1244	
S8	9	8	26	63	760	1471	1736	2496	
S9	10	9	29	75	926	1803	2132	3058	
S10	11	10	32	88	1108	2168	2568	3676	

Table 17.4 Number of qubits required for hypercubes and some other small known graphs

Graph	Order	Size	Integer variables	Quadratic constraints	Binary variables	Binary constraints	Slack variables	Logical qubits	Chimera qubits
Q1 = K2	2	1	5	7	9	15	13	22	47
Q2 = C4	4	4	13	21	74	131	146	220	662
Q3	8	12	33	65	447	851	999	1446	
Q4	16	32	81	193	2564	5045	5860	8424	
BidiakisCube	12	18	49	97	1432	2779	3124	4556	
Bull	5	5	16	28	178	324	366	544	3523
Butterfly	5	6	18	33	192	353	414	606	5927
Chvatal	12	24	61	145	1540	3013	3604	5144	
Clebsch	16	40	97	273	2708	5373	6628	9336	
Diamond	4	5	15	27	84	151	174	258	742
Dinneen	9	21	52	142	994	1950	2552	3546	
Dodecahedral	20	30	81	161	5515	10855	11645	17160	
Durer	12	18	49	97	1432	2779	3124	4556	
Errera	17	45	108	320	4480	8900	10890	15370	
Frucht	12	18	49	97	1432	2779	3124	4556	
GoldnerHarary	11	27	66	209	1414	2814	3792	5206	
Grotzsch	11	20	52	118	1288	2508	2968	4256	
Heawood	14	21	57	113	1894	3691	4100	5994	
Herschel	11	18	48	101	1252	2429	2800	4052	
Hexahedral	8	12	33	65	447	851	999	1446	
Hoffman	16	32	81	193	2564	5045	5860	8424	
House	5	6	18	32	192	353	414	606	4176
Icosahedral	12	30	73	205	1648	3257	4164	5812	
Krackhardt	10	18	47	114	1088	2116	2548	3636	
Octahedral	6	12	31	73	324	619	795	1119	
Pappus	18	27	73	145	4514	8869	9575	14089	
Petersen	10	15	41	81	1034	1995	2276	3310	
Poussin	15	39	94	276	2446	4863	6152	8598	
Robertson	19	38	96	229	5211	10287	11570	16781	
Shrikhande	16	48	113	369	2852	5715	7508	10360	
Sousselier	16	27	71	154	2474	4849	5452	7926	
Tietze	12	18	49	97	1432	2779	3124	4556	
Wagner	8	12	33	65	447	851	999	1446	

to exploit the problem's characteristics for a possible direct encoding into QUBO form. Also substantial work is needed to reduce the blowup in embedding the logical qubits to physical qubits, which is required for embedding into the target machine's architecture.

Acknowledgments This work was supported in part by the Quantum Computing Research Initiatives at Lockheed Martin. We thank A. Fowler for comments which improved the presentation.

Appendix: Quadratic IP Formulation for Broadcasting in Q_3

The output of our integer programming formulation from Sect. 17.5 with the hypercube Q_3 as input is given below.

$x_0 \leq 7$ (1) time t
$x_1 \leq x_0$ (2) vertices $v_0 \ldots v_7$ informed times $\leq t$
$x_2 \leq x_0$
$x_3 \leq x_0$
$x_4 \leq x_0$
$x_5 \leq x_0$
$x_6 \leq x_0$
$x_7 \leq x_0$
$x_8 \leq x_0$
$x_9 + x_{10} + x_{11} \leq 0$ (3) originator has no parent
$x_{12} + x_{13} + x_{14} \leq 1$ (4) other vertices have one parent
$x_{15} + x_{16} + x_{17} \leq 1$
$x_{18} + x_{19} + x_{20} \leq 1$
$x_{21} + x_{22} + x_{23} \leq 1$
$x_{24} + x_{25} + x_{26} \leq 1$
$x_{27} + x_{28} + x_{29} \leq 1$
$x_{30} + x_{31} + x_{32} \leq 1$
$x_{12} + x_{12} * x_1 - x_{12} * x_2 \leq 0$ (5) parent time less than child time
$x_{15} + x_{15} * x_1 - x_{15} * x_3 \leq 0$
$x_{21} + x_{21} * x_1 - x_{21} * x_5 \leq 0$
$x_9 + x_9 * x_2 - x_9 * x_1 \leq 0$
$x_{18} + x_{18} * x_2 - x_{18} * x_4 \leq 0$
$x_{24} + x_{24} * x_2 - x_{24} * x_6 \leq 0$
$x_{10} + x_{10} * x_3 - x_{10} * x_1 \leq 0$
$x_{19} + x_{19} * x_3 - x_{19} * x_4 \leq 0$
$x_{27} + x_{27} * x_3 - x_{27} * x_7 \leq 0$
$x_{13} + x_{13} * x_4 - x_{13} * x_2 \leq 0$
$x_{16} + x_{16} * x_4 - x_{16} * x_3 \leq 0$
$x_{30} + x_{30} * x_4 - x_{30} * x_8 \leq 0$
$x_{11} + x_{11} * x_5 - x_{11} * x_1 \leq 0$
$x_{25} + x_{25} * x_5 - x_{25} * x_6 \leq 0$
$x_{28} + x_{28} * x_5 - x_{28} * x_7 \leq 0$
$x_{14} + x_{14} * x_6 - x_{14} * x_2 \leq 0$
$x_{22} + x_{22} * x_6 - x_{22} * x_5 \leq 0$
$x_{31} + x_{31} * x_6 - x_{31} * x_8 \leq 0$
$x_{17} + x_{17} * x_7 - x_{17} * x_3 \leq 0$
$x_{23} + x_{23} * x_7 - x_{23} * x_5 \leq 0$
$x_{32} + x_{32} * x_7 - x_{32} * x_8 \leq 0$
$x_{20} + x_{20} * x_8 - x_{20} * x_4 \leq 0$
$x_{26} + x_{26} * x_8 - x_{26} * x_6 \leq 0$

$$x_{29} + x_{29} * x_8 - x_{29} * x_7 \leq 0$$
$$x_{12} + x_{15} - sqr(x_2 - x_3) \leq 1 \quad \text{(6) each child with different times}$$
$$x_{12} + x_{21} - sqr(x_2 - x_5) \leq 1$$
$$x_{15} + x_{21} - sqr(x_3 - x_5) \leq 1$$
$$x_9 + x_{18} - sqr(x_1 - x_4) \leq 1$$
$$x_9 + x_{24} - sqr(x_1 - x_6) \leq 1$$
$$x_{18} + x_{24} - sqr(x_4 - x_6) \leq 1$$
$$x_{10} + x_{19} - sqr(x_1 - x_4) \leq 1$$
$$x_{10} + x_{27} - sqr(x_1 - x_7) \leq 1$$
$$x_{19} + x_{27} - sqr(x_4 - x_7) \leq 1$$
$$x_{13} + x_{16} - sqr(x_2 - x_3) \leq 1$$
$$x_{13} + x_{30} - sqr(x_2 - x_8) \leq 1$$
$$x_{16} + x_{30} - sqr(x_3 - x_8) \leq 1$$
$$x_{11} + x_{25} - sqr(x_1 - x_6) \leq 1$$
$$x_{11} + x_{28} - sqr(x_1 - x_7) \leq 1$$
$$x_{25} + x_{28} - sqr(x_6 - x_7) \leq 1$$
$$x_{14} + x_{22} - sqr(x_2 - x_5) \leq 1$$
$$x_{14} + x_{31} - sqr(x_2 - x_8) \leq 1$$
$$x_{22} + x_{31} - sqr(x_5 - x_8) \leq 1$$
$$x_{17} + x_{23} - sqr(x_3 - x_5) \leq 1$$
$$x_{17} + x_{32} - sqr(x_3 - x_8) \leq 1$$
$$x_{23} + x_{32} - sqr(x_5 - x_8) \leq 1$$
$$x_{20} + x_{26} - sqr(x_4 - x_6) \leq 1$$
$$x_{20} + x_{29} - sqr(x_4 - x_7) \leq 1$$
$$x_{26} + x_{29} - sqr(x_6 - x_7) \leq 1$$

References

1. Tee, G.J.: The Monte Carlo Method. Pergamon Press, Oxford and New York (1966)
2. Metropolis, N.C., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E.: J. Chem. Phys. **21**(6), 1087 (1953)
3. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Science **220**(4598), 671 (1983)
4. Wikipedia, Simulated annealing. http://en.wikipedia.org/wiki/Simulated_annealing (2014). Accessed 30 Oct 2014
5. Born, M., Fock, V.: Zeitschrift für Physik **51**(3–4), 165 (1928). doi:10.1007/BF01343193
6. Farhi, E., Goldstone, J., Gutmann, S., Sipser, M.: Quantum computation by adiabatic evolution (2000). arXiv:quant-ph/0001106v1
7. Aharonov, D., Dam, W.v., Kempe, J., Landau, Z., Lloyd, S., Regev, O.: Adiabatic quantum computation is equivalent to standard quantum computation (2005). arXiv:quant-ph/0405098
8. D-Wave. D-Wave overview: a brief introduction to D-Wave and quantum computing (2013). http://www.dwavesys.com/sites/default/files/D-Wave-brochure-102013F-CA.pdf
9. Farley, A., Hedetniemi, S., Mitchell, S., Proskurowski, A.: Discret. Math. **25**, 189 (1979)
10. Hedetniemi, S.M., Hedetniemi, S.T., Liestman, A.L.: Networks **18**, 319 (1998)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., New York (1979)

12. Dinneen, M.J.: The complexity of broadcasting in bounded-degree networks. Technical Report Combinatorics report LACES-[05C-94-31], Los Alamos National Laboratory (1994). arXiv:math/9411222
13. Ravi, R.: In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, FOCS'94, pp. 202–213. IEEE Computer Society Press (1994)
14. Wikipedia, Integer programming. http://en.wikipedia.org/wiki/Integer_programming (2015). Accessed 30 Oct 2014
15. Khosravani, M.: Searching for optimal caterpillars in general and bounded treewidth graphs. Ph.D. dissertation, University of Auckland, Auckland, New Zealand (2011)
16. Stein, W., et al.: Sage Mathematics Software (Version 6.3). The Sage Development Team (2014). http://www.sagemath.org
17. Calude, C.S., Dinneen, M.J.: Solving the broadcast time problem using a D-wave quantum computer. Tech. Rep. Report CDMTCS-473, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, Auckland, New Zealand (2014). Paper URL http://www.cs.auckland.ac.nz/research/groups/CDMTCS/researchreports/index.php?download&paper_file=526. Data URL http://www.cs.auckland.ac.nz/research/groups/CDMTCS/researchreports/index.php?download&data_file=7
18. D-Wave, Programming with QUBOs. Technical report, D-Wave Systems, Inc. (2013). Python Release 1.5.1-beta4 (for Mac/Linux), 09-1002A-B