CDMTCS
Research
Report
Series

Finding the Chromatic Sums of Graphs Using a D-Wave Quantum Computer

Anuradha Mahasinghe
Department of Mathematics,
University of Colombo,
Sri Lanka

Michael J. Dinneen
Kai Liu
Department of Computer Science,
University of Auckland,
Auckland, New Zealand

CDMTCS-519
February 2018

Centre for Discrete Mathematics and
Theoretical Computer Science

Finding the Chromatic Sums of Graphs Using a D-Wave Quantum Computer

Anuradha Mahasinghe, Michael J. Dinneen and Kai Liu

Abstract

In this paper we demonstrate how to solve the chromatic sum problem using a D-Wave quantum computer. Starting from a BIP (binary integer programming) formulation, we develop a D-Wave feasible QUBO (quadratic unconstrained binary optimisation) formulation of the chromatic sum problem. Our construction requires nk qubits for a graph of n vertices and upper bound of k colours. Further, we present the experimental results obtained by running several QUBOs on a D-Wave quantum computer.

keywords: chromatic sum; graph colouring; QUBO formulation; adiabatic quantum computing.

1 Introduction

The graph theoretic concept of the chromatic sum of a graph was introduced by Kubicka and Schwenk in 1989 [25] and it has been suggested in 1987 from an application perspective by Supowit in [33]. Given a legal vertex colouring of a graph, it is possible to represent each colour by a positive integer, and to take the summation over all vertices. The minimum possible value for this summation is called the *chromatic sum* of the graph. Since its inception, the concept of the chromatic sum has gained the attention of mathematicians and computer scientists due to its computational hardness and application aspects.

As the well-known chromatic index (vertex colouring) problem, the chromatic sum problem too is NP–complete. Given an arbitrary graph, finding a valid assignment of colours (having adjacent vertices with different colours) which minimises the total summation of associated colours over the vertex set is proven to be NP–complete [25]. However, it has been proven that the problem is solvable in polynomial time for trees [25], unicyclic graphs [26] and outerplanar graphs [24, 26]. Finding the chromatic sum for arbitrary graphs has also been paid attention and the results indicate it is harder than the chromatic index problem for several classes of graphs [32, 23, 19]. Certain theoretical bounds for the problem were derived in [11, 15, 34, 29]. Different computational results were presented in [27, 4, 3, 10]. A descriptive survey of chromatic sum algorithms and results can be found in [20].

In the last decade, the paradigm of quantum computing emerged, dragging the attention of a wide-ranging community, including computer scientists, physicists, mathematicians and engineers. Though quantum computers have provided exponentially faster solutions to several problems and led to a diverse range of applications, originally it was not aimed at providing efficient solutions to NP–complete problems. The particular quantum computational framework of *adiabatic computing* was introduced in [13], in order to circumvent this issue. Though it is still debatable whether quantum computers can solve NP–complete problems in polynomial time, algorithms designed upon the adiabatic framework have provided efficient solutions to several instances of NP–complete problems [14, 12, 7].

The adiabatic framework of quantum computing gained a significant distinction after the invention of the D-Wave quantum computer. Despite the fact that the D-Wave machine is not a universal quantum computer, it had been capable of generating efficient and accurate solutions to several computational problems [5, 7]. The D-Wave quantum computer can be considered a hardware heuristic which minimises an unconstrained objective function using a physically realised version of a metaheuristic called *quantum annealing*, which is designed over the adiabatic framework [22]. One form of problems the D-Wave quantum computer claims to solve efficiently by quantum annealing is the class of QUBO (quadratic unconstrained binary optimisation) problems [9]. In previous works, several graph problems were converted into QUBO or similar optimisation problems and tested on the D-Wave computer. Examples include graph isomorphism [7], vertex colouring [35], broadcast-time [6] and spanning tree calculation [31] problems.

In this work, we solve the chromatic sum problem using a D-Wave quantum computer. We first restate the problem as a D-Wave feasible QUBO formulation and then run it on a D-Wave computer, and finally present our results with classically obtained verifications. Accordingly, the paper is organised as follows: Section 1 contains the introduction. In Section 2, we present necessary definitions and describe the chromatic sum problem. Section 3 contains the QUBO formulation with its proof of correctness. Section 4 contains our experimental results and Section 5 finishes with a discussion.

2 Chromatic Sum Problem

Given an undirected graph $G = (V, E)$ with a vertex set V and an edge set E, a *legal colouring* of its vertices could be considered as a mapping of its vertex set to natural numbers so that no two adjacent vertices are mapped into the same number. More precisely, a legal colouring is a mapping $c : V \to \mathbf{N}$ satisfying $c(u) \neq c(v)$ for all $(u, v) \in E$. Let $\Gamma(G)$ denote the class of all legal colourings of the graph G. Then the *chromatic sum* $\Sigma(G)$ of the graph G is defined as $\Sigma(G) = \min\{\sum_{v \in V} c(v) : c \in \Gamma(G)\}$.

A closely related problem is the problem of colourability or the chromatic index problem, which has been studied extensively in graph theory and computer science. That is, finding the legal colouring of the graph G with smallest number of different colours. The smallest number of colours necessary for a legal colouring of some graph G is called the *chromatic index* of that graph and it is denoted by $\chi(G)$. It is straightforward to see that $\Sigma(G) \geq \chi(G)$.

The chromatic index problem is a well-known candidate in the list of NP–complete problems [21]. It is straightforward to see that the chromatic sum problem is different from the

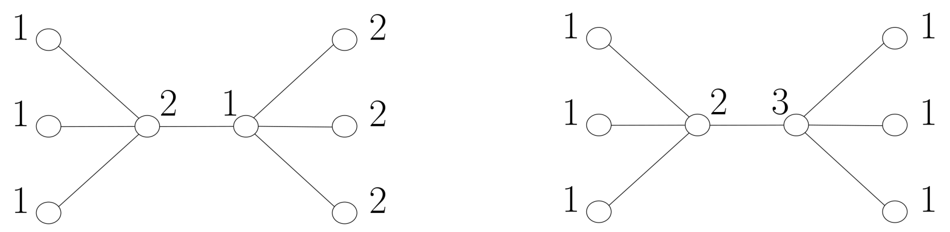

Figure 2.1: A tree T and two legal colourings.

chromatic index problem, as a colouring which would be optimal for the chromatic index problems might not be optimal for the chromatic sum problem. This is illustrated in the next example, which has been used to describe the notion of chromatic sum in [25].

2.1 Example

Consider the tree T in Figure 2.1 on 8 vertices with 6 leaves, for which a legal colouring requires only two colours. The minimum summation of colours over the vertex of set of this tree for such a colouring is $\sum_{v \in V} c(v) = 12$, as depicted in the colouring on the right of the figure. In order to minimise the summation of colours over the vertex set, it is necessary to use three colours, though a legal colouring requires only two colours. This is depicted in the colouring on the left of the figure. For this colouring, $\sum_{v \in V} c(v) = 11$, which is the optimal and hence $\Sigma(T) = 11$. Thus, the chromatic sum problem and the vertex colouring problem are not equivalent to each other.

2.2 An application

The chromatic sum problem has a direct application in distributed resource allocation [1]. Assume the vertices represent processors which execute tasks, using resources. If two processors require the same resource, they cannot operate at the same time and one processor may have to wait until the other accomplishes the task. Suppose the execution time is constant for each job. Then a processor to which the jth colour is assigned waits for $j - 1$ time units to receive the resources. In order to minimise the total waiting time of the system, one may need to find the optimal colour assignment, which gives the chromatic sum, when the colours are labelled from 0 to $k - 1$. It is straightforward to see that a relabelling from 1 to k with a shift of 1 for every vertex does not alter the optimal colour assignment. Thus, this application of resource allocation is equivalent to the chromatic sum problem.

In a more realistic message passing setting, the assumption that each task requires a constant time can be lifted. This results in a slightly different problem; and it has been the motivation for VLSI design [30]. Also several other problems were derived from the chromatic sum problem, such as the drinking philosophers problem [8], optimum cost chromatic partition problem [18] and sum multi-colouring problem [2].

3 QUBO Formulation

In order to restate the chromatic sum problem in a D-Wave feasible format, it is important to convert it to a QUBO problem. However, not all QUBOs are efficiently solvable by the D-Wave computer. Therefore, the conversion to final QUBO is a non-trivial task, in which the slack variables must be avoided and sparsity conditions must be paid attention. On the other hand, formulating the chromatic sum problem as a binary optimisation problem is straightforward. Thus, we first write it as a Binary Integer Programming (BIP) problem, with the intention of converting it to a QUBO problem of the required format.

3.1 BIP formulation

Given a graph $G = (V, E)$ on n vertices such that $V = \{v_0, v_1, \ldots, v_{n-1}\}$ and k colours ($\chi(G) \leq k \leq |V|$), we define the binary variable $x_{i,j}$ where $0 \leq i \leq n-1$ and $1 \leq j \leq k$, as follows.

$$x_{i,j} = \begin{cases} 1 & \text{if vertex } i \text{ is assigned colour } j, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, we define a binary vector $\mathbf{x} \in \mathbb{Z}_2^{nk}$ of decision variables for our formulation. It is straightforward to see that the function to be minimised can be written as

$$G(\mathbf{x}) = \sum_{i=0}^{n-1} \sum_{j=1}^{k} j x_{i,j}, \tag{1}$$

subject to the constraints,

$$\sum_{j=1}^{k} x_{i,j} = 1 \text{ for all } i \in \{0, 1, \ldots, n-1\}, \tag{2}$$

and

$$x_{i_1,j} + x_{i_2,j} \leq 1 \text{ for all } (v_{i_1}, v_{i_2}) \in E. \tag{3}$$

The constraint given by Equation (2) guarantees that each vertex is assigned a unique colour. The condition of legality of the colouring is imposed as the constraint in Equation (3). This BIP formulation is very similar to that of the standard chromatic index problem, and it has been presented previously in [32].

3.2 QUBO formulation

Motivated from the BIP formulation, we now formulate a D-Wave feasible QUBO for the chromatic sum problem. Although there is a generic conversion of BIP into QUBO, we do not adopt this method, as including slack variables or redundant decision variables is proven to affect the optimality of D-Wave solutions [5].

QUBO is an NP–hard mathematical problem of minimising a quadratic objective function $\mathbf{x}^\mathbf{T} Q\mathbf{x}$, where $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ is a vector of binary variables and Q is a diagonal symmetric matrix. In this work, all QUBOs considered are of the following form.

$$x^* = \min_{\mathbf{x}} \sum x_i Q_{(i,j)} x_j, \text{where } x_i, x_j \in \{0{,}1\}. \tag{4}$$

Now we state the QUBO objective function to be minimised in the chromatic sum problem.

$$F(\mathbf{x}) = \sum_{i=0}^{n-1} \sum_{j=1}^{k} j x_{i,j}^2 + A\left(P_1(\mathbf{x}) + P_2(\mathbf{x})\right) \tag{5}$$

where,

$$P_1(\mathbf{x}) = \sum_{i=0}^{n-1} \left(1 - \sum_{j=1}^{k} x_{i,j}\right)^2 \tag{6}$$

and

$$P_2(\mathbf{x}) = \sum_{(i_1,i_2)\in E} \sum_{j=1}^{k} x_{i_1,j} x_{i_2,j}. \tag{7}$$

Once the optimal solution of the objective function is obtained, the chromatic sum is given by $F(\mathbf{x}^*)$ and the respective optimal colour assignment by $\mathbf{x}^*$.

The term $\sum_{i=0}^{n-1} \sum_{j=1}^{k} j x_{i,j}^2$ in the objective function sums up the numbers used for colouring V while P_1 and P_2 serve as a penalties. This summation term is a simply the quadratic version of the linear objective function (Equation 1) in the BIP formulation in Section 3.1, since for all variables $x_i \in \{0, 1\}$ we have $x_i = x_i^2$. The term P_1 penalises if some vertex is coloured by more than one colour or not coloured by any as follows: If some vertex i in V is assigned with more than one colour, we have $\sum_{j=1}^{k} x_{i,j} > 1$, thus $1 - \sum_{j=1}^{k} x_{i,j} \neq 0$. Similarly, if a vertex is not coloured then also $1 - \sum_{j=1}^{k} x_{i,j} \neq 0$. Last term P_2 in the QUBO objective function penalises whenever the colouring is illegal. That is, for any pair of vertices $(u, v) \in E$, if they share the same colour j, we have $x_{u,j} = x_{v,j} = 1$. Since all variables are binary, $x_{u,j} x_{v,j} > 0$. A penalty scalar A is finally applied to P_1 and P_2, according to the fact that $A \geq n$ is sufficient for this formulation to be correct.

3.3 Proof of correctness

Now we establish the precise relation between the chromatic sum problem and our QUBO problem.

Given a variable assignment $\mathbf{x}$ for the QUBO, it is possible to define the corresponding relation $c_\mathbf{x} : V \to \mathbf{N}$ as $c_\mathbf{x}(v_i) = j$ if and only if $x_{i,j} = 1$. This becomes a legal colour scheme of vertices due to the constraints in the formulations. Once the optimal variable assignment for $F(\mathbf{x})$ is done, we get the corresponding colour scheme with the chromatic sum. In order to prove this, we first need to show that the penalties in Equation (6) and Equation (7) ensure a legal vertex colouring of the graph.

Lemma 1. *$c_{\mathbf{x}}$ is a legal vertex colouring if and only if $P_1(\mathbf{x}) = P_2(\mathbf{x}) = 0$.*

Proof. Suppose the relation $c_{\mathbf{x}} : V \to \mathbf{N}$ is a legal vertex colouring. Then it is well defined and therefore for each $v_i \in V$, there exists a unique $j \in \mathbf{N}$ such that $c_{\mathbf{x}}(v_i) = j$. That is, given $i \in \{0, 1, \ldots, n-1\}$, there is one and only one j satisfying $c_{\mathbf{x}}(v_i) = j$ or equivalently $x_{i,j} = 1$. Therefore, for each $i \in \{0, 1, \ldots, n-1\}$, $1 - \sum_{j=1}^{k} x_{i,j} = 1 - 1 = 0$. Therefore, $P_1(\mathbf{x}) = 0$. Secondly, since $c_{\mathbf{x}}$ is a legal vertex colouring, for all $i_1, i_2 \in \{0, 1, \ldots, n-1\}$, if $(v_{i_1}, v_{i_2}) \in E$ then $c_{\mathbf{x}}(v_{i_1}) \neq c_{\mathbf{x}}(v_{i_2})$. That is, given some j, not both $c_{\mathbf{x}}(v_{i_1})$ and $c_{\mathbf{x}}(v_{i_2})$ equal to j. This implies $x_{i_1,j} = 0$ or $x_{i_2,j} = 0$. From Equation (7) it follows that $P_2(\mathbf{x}) = 0$.

Now suppose $c_{\mathbf{x}} : V \to \mathbf{N}$ is not a legal vertex colouring. Then it is not well-defined or it assigns the same colour to some pair of adjacent vertices. Considering the case it is not well-defined, there exists some $i \in \{0, 1, \ldots, n-1\}$ and $j_1, j_2 \in \mathbf{N}$ such that $x_{i,j_1} = x_{i,j_2} = 1$. That is, $1 - \sum_{j=1}^{k} x_{i,j} \neq 0$, which implies $P_1(\mathbf{x}) \neq 0$. On the other hand, if $c_{\mathbf{x}}$ assigns the same natural number to some pair of adjacent vertices, then there exists some $i_1, i_2 \in \{0, 1, \ldots, n-1\}$ and $j \in \mathbf{N}$ such that $x_{i_1,j} = x_{i_2,j} = 1$, making $P_2(\mathbf{x}) \neq 0$. $\square$

Theorem 2. *$\mathbf{x}^*$ is an optimal variable assignment of the objective function $F(\mathbf{x})$ if and only if $c_{\mathbf{x}^*}$ is a legal colouring of vertices such that $F(\mathbf{x}^*) = \Sigma(G)$.*

Proof. First suppose that $c_{\mathbf{x}^*}$ is a legal colouring of vertices with $F(\mathbf{x}^*) = \Sigma(G)$. From Lemma 1, $P_1(\mathbf{x}^*) = P_2(\mathbf{x}^*) = 0$. Then, from Equation (5) it follows that

$$F(\mathbf{x}^*) = \sum_{i=0}^{n-1} \sum_{j=1}^{k} j {x^*_{i,j}}^2 \quad .$$

Therefore, $\Sigma(G) = \sum_{i=0}^{n-1} \sum_{j=1}^{k} j {x^*_{i,j}}^2$. Assume $\mathbf{x}^{**}$ to be an optimal variable assignment of the objective function $F(\mathbf{x})$. Then, by the definition of the chromatic sum, $\Sigma(G) \leq \sum_{i=0}^{n-1} \sum_{j=1}^{k} j {x^{**}_{i,j}}^2$. Combining these relations together, we have $F(\mathbf{x}^{**}) = \sum_{i=0}^{n-1} \sum_{j=1}^{k} j {x^{**}_{i,j}}^2 \geq \Sigma(G) = F(\mathbf{x}^*)$. Therefore, $\mathbf{x}^*$ is an optimal variable assignment.

Now consider the case where $c_{\mathbf{x}^*}$ is not a legal colouring of vertices. Also consider the colouring $c_{\mathbf{y}^*}$, which corresponds to the chromatic sum of the graph. Since $c_{\mathbf{y}^*}$ must be a legal colouring, by Lemma 1, $P_1(\mathbf{y}^*) + P_2(\mathbf{y}^*) = 0$. Further, by Equation (5), $F(\mathbf{y}^*) = \sum_{i=0}^{n-1} \sum_{j=1}^{k} j {y^*_{i,j}}^2 = \Sigma(G)$. Now, we consider a step-by-step procedure of forming the colouring $c_{\mathbf{x}^*}$ from $c_{\mathbf{y}^*}$. That is, at each step, we consider a single vertex of the graph coloured according to $c_{\mathbf{y}^*}$ and check its colour in $c_{\mathbf{x}^*}$. If it is different, we change the colour of the vertex; keep it unchanged otherwise. Further, at each step, we update the corresponding objective functions in Equation (5).

If the colour of the relevant vertex is changed at some step of this process, the corresponding term $\sum_{i=0}^{n-1} \sum_{j=1}^{k} j {y^*_{i,j}}^2$ in the objective function can be reduced by at most $n-1$. On the other hand, this increases the term $P_1(\mathbf{y}^*) + P_2(\mathbf{y}^*)$ by at least 1. Recall we select $A \geq n$, the term $A(P_1(\mathbf{y}^*) + P_2(\mathbf{y}^*))$ is increased by at least n due to this. Thus, the objective function in Equation (5) is increased at least by 1, if the colour of the vertex is changed at a single step. Applying this repeatedly, we end up with the colouring $c_{\mathbf{x}^*}$ and the updated

Table 1: Chromatic sum QUBO matrix for the tree T of Figure 2.1 with $k = 3$.

variables	$x_{0,1}$	$x_{0,2}$	$x_{0,3}$	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	$x_{7,1}$	$x_{7,2}$	$x_{7,3}$
$x_{0,1}$	-7	8	8	8	0	0	8	0	0	8	0	0	8	0	0	0	0	0	0	0	0	0	0	0
$x_{0,2}$	8	-6	8	0	8	0	0	8	0	0	8	0	0	8	0	0	0	0	0	0	0	0	0	0
$x_{0,3}$	8	8	-5	0	0	8	0	0	8	0	0	8	0	0	8	0	0	0	0	0	0	0	0	0
$x_{1,1}$	8	0	0	-7	8	8	0	0	0	0	0	0	0	0	0	8	0	0	8	0	0	8	0	0
$x_{1,2}$	0	8	0	8	-6	8	0	0	0	0	0	0	0	0	0	0	8	0	0	8	0	0	8	0
$x_{1,3}$	0	0	8	8	8	-5	0	0	0	0	0	0	0	0	0	0	0	8	0	0	8	0	0	8
$x_{2,1}$	8	0	0	0	0	0	-7	8	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{2,2}$	0	8	0	0	0	0	8	-6	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{2,3}$	0	0	8	0	0	0	8	8	-5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$x_{3,1}$	8	0	0	0	0	0	0	0	0	-7	8	8	0	0	0	0	0	0	0	0	0	0	0	0
$x_{3,2}$	0	8	0	0	0	0	0	0	0	8	-6	8	0	0	0	0	0	0	0	0	0	0	0	0
$x_{3,3}$	0	0	8	0	0	0	0	0	0	8	8	-5	0	0	0	0	0	0	0	0	0	0	0	0
$x_{4,1}$	8	0	0	0	0	0	0	0	0	0	0	0	-7	8	8	0	0	0	0	0	0	0	0	0
$x_{4,2}$	0	8	0	0	0	0	0	0	0	0	0	0	8	-6	8	0	0	0	0	0	0	0	0	0
$x_{4,3}$	0	0	8	0	0	0	0	0	0	0	0	0	8	8	-5	0	0	0	0	0	0	0	0	0
$x_{5,1}$	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	-7	8	8	0	0	0	0	0	0
$x_{5,2}$	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	8	-6	8	0	0	0	0	0	0
$x_{5,3}$	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	8	8	-5	0	0	0	0	0	0
$x_{6,1}$	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-7	8	8	0	0	0
$x_{6,2}$	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	8	-6	8	0	0	0
$x_{6,3}$	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	8	8	-5	0	0	0
$x_{7,1}$	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-7	8	8
$x_{7,2}$	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	-6	8
$x_{7,3}$	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	8	-5

objective function equal to $F(\mathbf{x}^*)$, which is strictly greater than the original objective function $F(\mathbf{y}^*)$, since $c_{\mathbf{x}^*}$ is not a legal colouring of vertices. That is, $F(\mathbf{x}^*) > F(\mathbf{y}^*)$, proving the non-optimality of $\mathbf{x}^*$.

Further, if $c_{\mathbf{x}^*}$ is a legal colouring of vertices with $F(\mathbf{x}^*) \neq \Sigma(G)$, by definitions it follows straight away that $F(\mathbf{x}^*) > \Sigma(G)$. Considering again the colouring $c_{\mathbf{y}^*}$, which gives $\Sigma(G)$, non-optimality of $\mathbf{x}^*$ is proven. $\square$

3.4 Example: the tree T revisited

Consider the example tree T in Figure 2.1. The vertex set of T is given by $V = \{0, 1, \ldots, 7\}$ and the edge set by $E = \{(0,1), (0,2), (0,3), (0,4), (1,5), (1,6),$
$(1,7)\}$. As described in 3.2, all non-quadratic terms are removed, all constant terms are ignored, and all linear terms are replaced by the squares of those terms. With the selection A = 8, we compute the coefficients of the each quadratic term and generate the matrix form in Table 1.

The variable assignment of our previous two solutions in Figure2.1:

$$\mathbf{x}_a = [0,1,0,0,0,1,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0]$$
$$\mathbf{x}_b = [0,1,0,0,0,1,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,1,0,0]$$

Decoding $\mathbf{x}_a$ gives a legal colouring $c_{\mathbf{x}_a} = \{(0,2), (1,3), (2,1), (3,1), (4,1), (5,1),$
$(6,1), (7,1)\}$ with the optimal solution $\sum c_{\mathbf{x}_a}(\mathbf{v}) = 11$ and $\mathbf{x}_b$ gives a valid colouring $c_{\mathbf{x}_b} = \{(0,2), (1,1), (2,1), (3,1), (4,1), (5,2), (6,2), (7,2)\}$ with the summation $\sum c_{\mathbf{x}_b}(\mathbf{v}) = 12$.

4 Computational Results

Given a QUBO, the binary decision variable x_i can be directly transformed into a spin decision variable s_i, through the simple transformation $s_i = 1 - 2x_i$. Thus, the spin variable s_i takes the value 1 or -1 as x_i is 0 or 1. These spin variables form a physical Hamiltonian, which is the key ingredient in D-Wave quantum computer [28]. Thus, we say the decision variable x_i represents the ith *logical* qubit of the problem. Considering the physical process, the Hamiltonian can be stated in the form of interactions between the ith and jth qubits of the physical system which represents the relation between x_i and x_j of the original problem. In adiabatic computing, an initial Hamiltonian which is easy to prepare undergoes a sufficiently slow process to become the problem Hamiltonian mentioned above, giving the solutions of the problem. The D-Wave machine is a hardware heuristic which uses a metaheuristic resembling this adiabatic process.

We tested our QUBO formulation of the chromatic on a D-Wave quantum computer for several graphs. We used a D-Wave 2X computer which consisted of 1098 active *physical* qubits. The physical qubit architecture of D-Wave 2X computer is a Chimera graph, which makes an embedding on the hardware for our QUBO instance a necessity, before we run any QUBO instance on it.

We experimented 14K trials on each graph with and without D-Wave post-processing optimisation. The experimental results are summarised in Table 2. Columns *order* and *size* indicate the number of vertices and edges of graphs. The column *chromatic number* indicates the smallest number of colours needed for a valid vertex colouring and *selection of k* is the k for our QUBO formulation of chromatic sum in Equation 5. Results in Table 2 are generated by assigning $k = \chi(G) + 2$. The column *logical qubits* contains the number of variables of the QUBO formulation while *physical qubits* gives the number of actual qubits working on the D-Wave computer after embedding. The column *embedding max chain* is the maximum number of physical qubits to which a single logical qubit is mapped. The next two columns *minimum without postproc optimise* and *minimum postproc optimise* show the minimal sum of colouring from the 14K trials with and without post-processing and *minimum D-Wave* takes the minimum of them. In these three columns, if a cell is filled by a dash, it means no feasible colouring is ever found out of the 14K trials. The cells highlighted with red colour are the minimum sums of colours D-Wave found which provided non-optimal answers.

In order to verify the results obtained by D-Wave computer, we generated solutions for the chromatic sum problem in a conventional setting as well. Accordingly, we solved the BIP formulation in Section 3.1 using the CPLEX BIP solver [17]. Further, we generated solutions to our QUBOs using CPLEX. The column *optimal answer* in Table 2 corresponds to these computational results in a conventional setting. Our CPLEX QUBO solver takes QUBO instances generated in Appendix A and computes the results. The programs used to submit, solve and verify on the D-Wave machine are given in Appendices B and C. The solver programs written in C++ using CPLEX concert technology are available in Appendices D and E.

For example, consider the tree T in Figure 2.1, with the selection $k = 4$. This graph with 8 vertices and 7 edges is listed as the last item in Table 2. From Figure 2.1 we see that the graph is colourable by 2 colours and its chromatic index $\chi(T) = 2$. Our selection for k is $\chi(T) + 2 = 4$. According to the relevant QUBO formulation, the problem contains 32 binary variables (Table 1 gives the QUBO matrix for $k = 3$). Accordingly, in order to form the physical Hamiltonian for the adiabatic quantum computation in the D-Wave architecture, we need 32 logical qubits. Due to embedding considerations, the D-Wave topology requires 79 physical qubits in its hardware. With and without post processing optimisation option, the D-Wave gives the expected answer 11, which is given in respective columns. A classically verified optimal answer is given in the last column.

5 Discussion

The NP–complete problem of the chromatic sums of graphs has been addressed previously in several computational frameworks, in particular using different heuristics. Examples include greedy algorithms [27], ant-colony optimisation [10], tabu search [4] and local search [16] for the chromatic sum problem. Though the heuristic approach has been improved significantly by recent authors, experimental results indicate that it could still be inefficient for several problem instances. Motivated from the fact that the metaheuristic approach of D-Wave has provided efficient and accurate solutions for many instances of graph theoretic problems, we experimented with the chromatic sum problem in that architecture and derived efficient results. Accordingly, we presented the chromatic sum of graphs, its QUBO formulation and computational solutions for QUBO, computed by a D-Wave 2X computer.

Recall the BIP formulation can be directly converted into a QUBO problem by a generic quadratic transformation. A QUBO formulation for the chromatic sum problem derived from the relevant BIP formulations were previously experimented in [10]. The authors have used genetic algorithms in their approach and reported that the CPU time to reach the smallest sum coloring is less than one minute for several instances. A similar quadratic formulation has been solved using a path relinking algorithm in [36]. However, in our setting, a quadratic transformation of the BIP formulation does not seem to be very useful, as the number of variables or the required logical qubits grows rapidly. A similar behaviour was noted also for the BIP formulation of the broadcast time problem [6]. The computational results indicate that although path relinking methods produce efficient answers for several instances, in general, it could take a significant time. The D-Wave solutions produced through our QUBO formulations were all efficient using only $O(nk)$ qubits, and with significant accuracy.

There are several graphs for which non-optimal solutions were generated. All these graphs require a higher number of physical qubits, though the number of required logical qubits was optimised by our QUBO formulation. More precisely, *embedding max chain* which corresponds to the maximum number of physical qubits to which a single logical qubit is mapped were above 10 for all these instances. This non-optimality could be considered a consequence of a non-optimal embedding. Previous works have reported a performance improvement due to reduced embedding max chain for other graph problems over D-Wave architecture such as graph covering [9] and graph isomorphism [7]. Our results too indicate a similar behaviour. It would be an interesting topic for further study to find an optimal embedding for the chromatic sum problem instances.

Table 2: Results for some graphs families for the chromatic sum problem.

Graph	Order	Size	Chromatic Number	Selection of k	Logical Qubits	Physical Qubits	Embedding Max Chain	Minimum Without PostProc Optimise	Minimum PostProc Optimise	Minimum D-Wave	Optimal Answer
BidiakisCube	12	18	3	5	60	392	11	28	23	23	21
Bull	5	5	3	5	25	85	4	8	8	8	8
Butterfly	5	6	3	5	25	91	5	9	9	9	9
C4	4	4	2	4	16	40	3	6	6	6	6
C5	5	5	3	5	25	81	5	9	9	9	9
C6	6	6	2	4	24	76	4	9	9	9	9
C7	7	7	3	5	35	168	6	12	12	12	12
C8	8	8	2	4	32	94	4	12	12	12	12
C9	9	9	3	5	45	164	5	15	15	15	15
C10	10	10	2	4	40	117	4	15	15	15	15
C11	11	11	3	5	55	209	7	20	18	18	18
C12	12	12	2	4	48	142	4	18	18	18	18
Diamond	4	5	3	5	20	90	6	7	7	7	7
Dinneen	9	21	3	5	45	378	14	21	19	19	18
Dodecahedral	20	30	3	5	100	660	11	49	42	42	38
Durer	12	18	3	5	60	342	9	27	24	24	24
Frucht	12	18	3	5	60	302	7	25	24	24	22
GoldnerHarary	11	27	4	6	66	660	21	-	30	30	22
Grid2x3	6	7	2	4	24	76	4	9	9	9	9
Grid3x3	9	12	2	4	36	97	4	13	13	13	13
Grid3x4	12	17	2	4	48	199	6	20	18	18	18
Grid4x4	16	24	2	4	64	333	11	31	24	24	24
Grid4x5	20	31	2	4	80	430	17	38	32	32	30
Grotzsch	11	20	4	6	66	729	19	27	28	27	21
Heawood	14	21	2	4	56	388	12	25	23	23	21
Herschel	11	18	2	4	44	216	7	16	16	16	16
Hexahedral	8	12	2	4	32	129	8	12	12	12	12
Hoffman	16	32	2	4	64	609	13	-	29	29	24
House	5	6	3	5	25	88	5	9	9	9	9
K2	2	1	2	4	8	14	2	3	3	3	3
K3	3	3	3	5	15	49	4	6	6	6	6
K4	4	6	4	6	24	134	8	10	10	10	10
K5	5	10	5	7	35	354	14	15	15	15	15
K6	6	15	6	8	48	592	17	-	21	21	21
K2,3	5	6	2	4	20	73	5	7	7	7	7
K3,3	6	9	2	4	24	107	7	9	9	9	9
K3,4	7	12	2	4	28	135	6	11	10	10	10
K4,4	8	16	2	4	32	173	8	12	12	12	12
K4,5	9	20	2	4	36	247	9	15	13	13	13
K2x1	3	2	2	4	12	21	2	4	4	4	4
K5x5	10	25	2	4	40	353	11	-	16	16	15
K5x6	11	30	2	4	44	418	14	-	19	19	16
K6x6	12	36	2	4	48	437	12	-	24	24	18
Krackhardt	10	18	4	6	60	416	12	29	23	23	20
Octahedral	6	12	3	5	30	195	8	12	12	12	12
Pappus	18	27	2	4	72	472	11	39	29	29	27
Petersen	10	15	3	5	50	303	10	20	19	19	19
Q3	8	12	2	4	32	138	6	12	12	12	12
Q4	16	32	2	4	64	601	13	-	30	30	24
S2	3	2	2	4	12	22	2	4	4	4	4
S3	4	3	2	4	16	34	3	5	5	5	5
S4	5	4	2	4	20	48	4	6	6	6	6
S5	6	5	2	4	24	66	5	7	7	7	7
S6	7	6	2	4	28	99	9	8	8	8	8
S7	8	7	2	4	32	84	6	11	9	9	9
S8	9	8	2	4	36	115	8	15	10	10	10
S9	10	9	2	4	40	122	9	18	11	11	11
S10	11	10	2	4	44	130	9	23	12	12	12
Sousselier	16	27	3	5	80	680	16	-	36	36	30
Tietze	12	18	3	5	60	427	12	28	22	22	22
Wagner	8	12	3	5	40	232	8	15	15	15	15
Tree_T	8	7	2	4	32	79	4	11	11	11	11

6 Acknowledgement

This work was supported in part by the Quantum Computing Research Initiatives at Lockheed Martin. AM would like to thank André Nies for helpful discussions and also for the funds received through the Department of Computer Science of the University of Auckland and Marsden grant.

References

[1] Amotz Bar-Noy, Mihir Bellare, Magnús M Halldórsson, Hadas Shachnai, and Tami Tamir. On chromatic sums and distributed resource allocation. *Information and Computation*, 140(2):183–202, 1998.

[2] Amotz Bar-Noy, Magnús M Halldórsson, Guy Kortsarz, Ravit Salman, and Hadas Shanhnai. Sum multi-coloring of graphs. In *European Symposium on Algorithms*, pages 390–401. Springer, 1999.

[3] Una Benlic and Jin-Kao Hao. A study of breakout local search for the minimum sum coloring problem. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 128–137. Springer, 2012.

[4] Hend Bouziri and Mouna Jouini. A tabu search approach for the sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36:915–922, 2010.

[5] Cristian S Calude, Elena Calude, and Michael J Dinneen. Guest column: Adiabatic quantum computing challenges. *ACM SIGACT News*, 46(1):40–61, 2015.

[6] Cristian S Calude and Michael J Dinneen. Solving the broadcast time problem using a D-Wave quantum computer. In *Advances in Unconventional Computing*, pages 439–453. Springer, 2017.

[7] Cristian S Calude, Michael J Dinneen, and Richard Hua. QUBO formulations for the graph isomorphism problem and related problems. *Theoretical Computer Science*, 701:54–69, 2017.

[8] K Mani Chandy and Jayadev Misra. The drinking philosophers problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 6(4):632–646, 1984.

[9] Michael J Dinneen and Richard Hua. Formulating graph covering problems for adiabatic quantum computers. In *Proceedings of the Australasian Computer Science Week Multiconference*, ACSW '17, pages 18:1–18:10, New York, NY, USA, 2017. ACM.

[10] Sidi Mohamed Douiri and Souad Elbernoussi. A new ant colony optimization algorithm for the lower bound of sum coloring problem. *Journal of Mathematical Modelling and Algorithms*, 11(2):181–192, 2012.

[11] Paul Erdös, Ewa Kubicka, and Allen J Schwenk. Graphs that require many colors to achieve their chromatic sum. *Congr. Numer*, 71:17–28, 1990.

[12] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. A quantum adiabatic evolution algorithm applied to random instances of an NP–complete problem. *Science*, 292(5516):472–475, 2001.

[13] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000.

[14] Silvano Garnerone, Paolo Zanardi, and Daniel A Lidar. Adiabatic quantum algorithm for search engine ranking. *Physical review letters*, 108(23):230506, 2012.

[15] Hossein Hajiabolhassan, Mojtaba L Mehrabadi, and Ruzbeh Tusserkani. Tabular graphs and chromatic sum. *Discrete mathematics*, 304(1-3):11–22, 2005.

[16] Anders Helmar and Marco Chiarandini. A local search heuristic for chromatic sum. In *Proceedings of the 9th metaheuristics international conference*, volume 1101, pages 161–170, 2011.

[17] IBM. ILOG CPLEX optimization studio CPLEX user's manual, 2017. https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer.

[18] Klaus Jansen. The optimum cost chromatic partition problem. In *Italian Conference on Algorithms and Complexity*, pages 25–36. Springer, 1997.

[19] Klaus Jansen. Approximation results for the optimum cost chromatic partition problem. *Journal of Algorithms*, 34(1):54–89, 2000.

[20] Yan Jin, Jean-Philippe Hamiez, and Jin-Kao Hao. Algorithms for the minimum sum coloring problem: a review. *Artificial Intelligence Review*, 47(3):367–394, 2017.

[21] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[22] James King, Sheir Yarkoni, Jack Raymond, Isil Ozfidan, Andrew D King, Mayssam Mohammadi Nevisi, Jeremy P Hilton, and Catherine C McGeoch. Quantum annealing amid local ruggedness and global frustration. *algorithms*, 5:7, 2017.

[23] Leo G Kroon, Arunabha Sen, Haiyong Deng, and Asim Roy. The optimal cost chromatic partition problem for trees and interval graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 279–292. Springer, 1996.

[24] Ewa Kubicka. The chromatic sum of a graph: History and recent developments. *International Journal of Mathematics and Mathematical Sciences*, 2004(30):1563–1573, 2004.

[25] Ewa Kubicka and Allen J Schwenk. An introduction to chromatic sums. In *Proceedings of the 17th conference on ACM Annual Computer Science Conference*, pages 39–45. ACM, 1989.

[26] Ewa M Kubicka. Polynomial algorithm for finding chromatic sum for unicyclic and outerplanar graphs. *Ars Combinatoria*, 76:193–202, 2005.

[27] Yu Li, Corinne Lucet, Aziz Moukrim, and Kaoutar Sghiouer. Greedy algorithms for the minimum sum coloring problem. In *Logistique et transports*, pages LT–027, 2009.

[28] Catherine C McGeoch. Adiabatic quantum computation and quantum annealing: Theory and practice. *Synthesis Lectures on Quantum Computing*, 5(2):1–93, 2014.

[29] Aziz Moukrim, Kaoutar Sghiouer, Corinne Lucet, and Yu Li. Lower bounds for the minimal sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36:663–670, 2010.

[30] Sara Nicoloso, Majid Sarrafzadeh, and X Song. On the sum coloring problem on interval graphs. *Algorithmica*, 23(2):109–126, 1999.

[31] Mark A Novotny, Lukas Hobl, JS Hall, and Kristel Michielsen. Spanning tree calculations on D-Wave 2 machines. In *Journal of Physics: Conference Series*, volume 681(1), page 012005. IOP Publishing, 2016.

[32] Arunabha Sen, Haiyong Deng, and Sumanta Guha. On a graph partition problem with application to vlsi layout. *Information processing letters*, 43(2):87–94, 1992.

[33] Kenneth J Supowit. Finding a maximum planar subset of a set of nets in a channel. *IEEE transactions on computer-aided design of integrated circuits and systems*, 6(1):93–94, 1987.

[34] Carsten Thomassen, Paul Erdös, Yousef Alavi, Paresh J Malde, and Allen J Schwenk. Tight bounds on the chromatic sum of a connected graph. *Journal of Graph Theory*, 13(3):353–357, 1989.

[35] Olawale Titiloye and Alan Crispin. Quantum annealing of the graph coloring problem. *Discrete Optimization*, 8(2):376–384, 2011.

[36] Yang Wang, Jin-Kao Hao, Fred Glover, and Zhipeng Lü. Solving the minimum sum coloring problem via binary quadratic programming. *arXiv preprint arXiv:1304.5876*, 2013.

A C++ program to generate QUBO

```
// QUBO formulation for chromatic sum problem
// usage: $./chromatic_sum_qubo k < graph.alist < graph.d.out
// k is set to be same as graph size n if not given by argument
#include <iostream>
#include <fstream>
#include <utility>
#include <list>
#include <sstream>
#include <string>
#include <stdlib.h>

using namespace std;
void read_graph(const int n, list <pair<int,int> > &adjacent_list);
void makeQuboMatrix(double **Q, int n, int k, list <pair<int,int> >
    adjacent_list);
void printMatrix(double **Q, const int n);

int main(int argc, char *argv[])
{
     int n=0,k=0;
     double **Q;
     list <pair<int,int> > adjacent_list;
     cin>>n;
     if (argc < 2) cout << "Correct usage: " << argv[0] <<" <filename>" << endl
    ;
     else if(argc == 2) k = (int)strtol (argv[1],NULL,10);
     else k = n;
     read_graph(n,adjacent_list);
     const int N = n*k;
     Q = new double*[N];
     for (int i=0; i<N; i++)
     {
         Q[i] = new double[N];
         for (int j=0; j<N; j++) Q[i][j] = 0;
     }

     makeQuboMatrix(Q,n,k,adjacent_list);
     cout<< n*k<<" "<<k<<endl;
     printMatrix(Q,N);
     for (int i=0; i<N; i++)
         delete [] Q[i];
     delete [] Q;
     return 0;
}

void read_graph(const int n, list <pair<int,int> > &adjacent_list)
{
  string line;
  int lineCnt=-1;
  for(int i=0;i<n+1;i++)
  {
    std::getline(cin, line);
    istringstream iss(line);
    int a;
    while (iss >> a)adjacent_list.push_back(make_pair(lineCnt,a));
    lineCnt++;
  }
}

void makeQuboMatrix(double **Q, int n, int k, list <pair<int,int> >
   adjacent_list)
{
    const int penalty = n;

    //f(x) = sum j*x[i,j]
    for (int i=0; i<n; i++)
    {
      for(int j=0; j<k; j++)
      {
          int idx = i*k+j;
         Q[idx][idx] = Q[idx][idx] + (j+1);
      }
    }
    //p1(x) = sum(i) (1- sum(j)x[i,j])^2
    for (int i=0; i<n; i++){
      for(int j=0; j<k; j++){
        for(int l=0; l<k; l++){
          if(j!=l){
            int idx1 = i*k+j;
            int idx2 = i*k+l;
            Q[idx1][idx2] += penalty;
          }
          else {
            int idx = i*k+j;
            Q[idx][idx] = Q[idx][idx] - penalty;
          }
        }
      }
    }

    //p2(x) = sum(E(G))sum(j) x[u,j]*x[v,j]
    list<pair<int,int> >::const_iterator iterator;

    for (iterator = adjacent_list.begin(); iterator != adjacent_list.end(); ++
   iterator) {
      int u = (*iterator).first;
      int v = (*iterator).second;
      for(int j=0; j<k; j++) {
        Q[u*k+j][v*k+j] += penalty;
      }
  }
}

void printMatrix(double **Q, const int n)
{
    for (int i=0; i<n; i++){
        for (int j=0; j<n; j++) cout<< Q[i][j]<<" ";
        cout <<endl;
    }
}
```

Listing 1: chromaticsum_qubo.cpp

B Python program to solve QUBO on D-Wave

```
#!/usr/bin/env python
# Chromatic Sum QUBO (with embedding) -> Ising -> DWave

import sys, time, math, traceback

from dwave_sapi2.remote import RemoteConnection
from dwave_sapi2.util import get_hardware_adjacency
from dwave_sapi2.embedding import embed_problem, unembed_answer
from dwave_sapi2.util import qubo_to_ising, ising_to_qubo
from dwave_sapi2.core import solve_ising

from sys import exc_info

# coupler streingth for embedded qubits of same variable

s,s2=1.0,1.0
print 'Embed scale=',s,s2
assert 0 <= s <= 1

# read input

line=sys.stdin.readline().strip().split()
n=int(line[0])

Q = {}
for i in range(n):
    line=sys.stdin.readline().strip().split()
    for j in range(n):
        t = float(line[j])
        if j>=i and t!=0: Q[(i,j)]=t

(H,J,ising_offset) = qubo_to_ising(Q)

# scale by maxV

maxH=0.0
if len(H): maxH=max(abs(min(H)),abs(max(H)))
maxJ=max(abs(min(J.values())),abs(max(J.values())))
maxV=max(maxH,maxJ)
for i in range(n):
    if len(H)>i:
        H[i]=s2*H[i]/maxV
    for j in range(n):
        if j>=i and (i,j) in J:
            J[(i,j)]=s2*J[(i,j)]/maxV

embedding=eval(sys.stdin.readline())
print 'embedding=', embedding
qubits = sum(len(embed) for embed in embedding)
print 'Physical qubits used= %s' % qubits

# create a remote connection using url and token and connect to solver

url = "dwave site"
token = "secret"
solver_name = "DW2X"

print('Attempting to connect to network...')
try:
    remote_connection = RemoteConnection(url, token)
    solver = remote_connection.get_solver(solver_name)
except:
    print('Error: %s %s %s' % sys.exc_info()[0:3])
    traceback.print_exc()

A = get_hardware_adjacency(solver)

# Embed problem into hardware

(h0, j0, jc, new_emb) = embed_problem(H, J, embedding, A)
h1= [val*s for val in h0]
j1 = {}
for (key, val) in j0.iteritems():
    j1[key]=val*s
j1.update(jc)
assert new_emb==embedding
(Q,offset) = ising_to_qubo(h1,j1)

# call the solver

annealT,progT,readT=20,100,100
print 'annealT=',annealT,'progT=',progT,'readT=',readT
result = solve_ising(solver, h1, j1, num_reads=10000, annealing_time=annealT,
         programming_thermalization=progT, readout_thermalization=readT,
         postprocess='optimization', num_spin_reversal_transforms=20)
print 'result:', result

newresult = unembed_answer(result['solutions'], new_emb, broken_chains='vote',
    h=H, j=J)
print 'newresult:', newresult
```

Listing 2: dwave_preembed.py

C C++ program to verify solutions from D-Wave

```
// C++ program for checking dwave answers
// usage : $./checkCSP graph.alist < graph.d.out
#include <iostream>
#include <fstream>
#include <list>
#include <sstream>
#include <string>
#include <vector>
using namespace std;
void read_graph(string filename, int &n, list <pair<int,int> > &adjacent_list)
    ;
vector<int*> ProcDwaveOutput(const int n, int &k);
void Verifysolution(vector<int*> solutions, int n, int k, list <pair<int,int>
    > adjacent_list);
void find_and_replace(string& source, string const& find, string const&
    replace);

int main(int argc, char *argv[])
{
    int n=0,k=0;
    list <pair<int,int> > adjacent_list;
    if (argc < 2) return 0;
    else if (argc==2)
      read_graph(argv[1], n, adjacent_list);

    vector<int*> solutions = ProcDwaveOutput(n, k);
    Verifysolution(solutions,n,k,adjacent_list);
    return 0;
}

void read_graph(string filename, int &n, list <pair<int,int> > &adjacent_list)
{
  ifstream infile( filename );
  if (infile.is_open()==0) exit(0);
  infile >> n;
  string line;
  int lineCnt=-1;
  for(int i=0;i<n+1;i++)
  {
    std::getline(infile, line);
    istringstream iss(line);
    int a;
    while (iss >> a)adjacent_list.push_back(make_pair(lineCnt,a));
    lineCnt++;
  }
}

vector<int*> ProcDwaveOutput(const int n, int &k){
  string line;
  vector<int*> solutions;
  while(std::getline(cin, line))
  {
    size_t kpos = line.find("k=");
    if (kpos!=string::npos && line.length()>=kpos+5)
    {
      string ks = line.substr(kpos,line.length()-kpos);
      ks = ks.substr(ks.find(",")+1, ks.find(")")-ks.find(",")-1);
      k = atoi(ks.c_str());
    }
    size_t rpos = line.find("newresult:");
    if (rpos!=string::npos){
      line = line.substr(line.find("[")+1);
      find_and_replace( line, "-1", "0");
      while(1){
        size_t pf = line.find("["), pe = line.find("]");
        if (pf!=string::npos && pe!=string::npos)
        {
          string s = line.substr(pf+1,pe-pf-1);
          line = line.substr(pe+1);
          solutions.push_back( new int[n*k] );
          for (int i=0;i<n*k;i++)solutions.back()[i] = s[i*3]-'0';
        }
        else if(pf==string::npos) break;
      }
    }
  }
  return solutions;
}

void Verifysolution(vector<int*> solutions, int n, int k, list <pair<int,int>
   > adjacent_list){
  const int N =n;
  int min=-1, gColoringNumber=0;
  for(int cnt=0;cnt<solutions.size();cnt++)
  {
    bool legal=true;
    int *x= solutions[cnt];
    int c[N],lColoringNumber=0;

    //check if every vertex is colored by exact one coloring number
    for (int i=0;i<n;i++)
    {
      int sumj=0;
      for(int j=0;j<k;j++)
      {
        sumj+=x[i*k+j];
        if(x[i*k+j] == 1)
        {
          c[i] = j+1;
          if (lColoringNumber < c[i]) lColoringNumber=c[i];
        }
      }
      if (sumj !=1 ) {legal = false;  break;}
    }
    if(legal==false) continue;

    //check if each pair of ajacent vertices are colored by different number
    list<pair<int,int> >::const_iterator iterator;
    for (iterator = adjacent_list.begin(); iterator != adjacent_list.end(); ++
   iterator) {
      int u = (*iterator).first, v = (*iterator).second;;
      if (c[u] == c[v]) {legal = false; break;}
    }

    if(legal==false) continue;
    int optimal=0;
    for (int i = 0; i < n; i++)
      for (int j = 0; j < k; j++) optimal = optimal + x[i*k+j] * (j+1);
    if(min==-1) {min = optimal; gColoringNumber = lColoringNumber;}
    else if(min>optimal) {min = optimal; if(gColoringNumber > lColoringNumber)
    gColoringNumber = lColoringNumber;}
  }

  if ( min!=-1) cout<<"Minimal sum of coloring found: "<<min<<endl<<"Number of
     color userd: "<<gColoringNumber<<endl;
  else cout<<"Optimize failed."<<endl;
}

void find_and_replace(string& source, string const& find, string const&
    replace)
{
    for(string::size_type i = 0; (i = source.find(find, i)) != string::npos;)
    {
        source.replace(i, find.length(), replace);
        i += replace.length();
    }
}
```

Listing 3: checkCSP.cpp

D CPLEX BIP Solver for Chromatic Sum Problem

```
// BIP solver for Chromatic Sum problem
// Usage: $./CSP_BIP_Solver k < graph.alist
// k is assigned to n if not given by argument
#include <ilcplex/ilocplex.h>
#include <iostream>
#include <fstream>
#include <list>
#include <sstream>
ILOSTLBEGIN
void read_graph(const int n, list <pair<int,int> > &adjacent_list);
int main (int argc, char **argv)
{
    int n=0,k=0;
    list <pair<int,int> > adjacent_list;
    cin>>n;
    if (argc == 2) k = (int)strtol (argv[1],NULL,10);
    else k=n;
    read_graph(n,adjacent_list);
    const int N = n*k;
    IloEnv   env;
    try {
        IloModel model(env);
        IloNumVarArray var(env);
        IloRangeArray con(env);

        for (int i = 0; i < N; ++i)
        {
            char buffer [10];
            sprintf (buffer, "x%d", i+1);
            var.add(IloNumVar(env,0,1,ILOBOOL,buffer));
        }

        IloExpr expr(env);
        //f(x) = sum j*x[i,j]
        for (int i=0; i<n; i++)
            for (int j=0; j<k; j++)
            {
                int idx = i*k+j;
                expr += var[idx]*(j+1);
            }
        IloObjective obj = IloMinimize(env,expr);

        IloExprArray expr_constrs = IloExprArray(env,n);
        for (int i=0; i<n; i++){
            expr_constrs[i] = IloExpr(env);
            for (int j=0; j<k; j++){
                int idx = i*k+j;
                expr_constrs[i] += var[idx];
            }
            char buffer [10];
            sprintf (buffer, "c%d", i+1);
            con.add(IloRange(env, 1,expr_constrs[i], 1,buffer));
        }

        //p2(x) = sum(E(G))sum(j) x[u,j]+x[v,j]<=1
        list<pair<int,int> >::const_iterator iterator;

        for (iterator = adjacent_list.begin(); iterator != adjacent_list.end()
   ; ++iterator) {
            int u = (*iterator).first;
            int v = (*iterator).second;
            for(int j=0; j<k; j++) con.add(var[u*k+j]+var[v*k+j]<=1);

        }

        model.add(obj);
        model.add(con);
        IloCplex cplex(model);
        cplex.exportModel("CSP_BIP_Solver.lp");
        cplex.setOut(env.getNullStream());
        // Optimize the problem and obtain solution.
        if ( !cplex.solve() ) {
            env.error() << "Failed to optimize LP" << endl;
            throw(-1);
        }

        IloNumArray vals(env);
        env.out() << "Solution status = " << cplex.getStatus() << endl;
        env.out() << "Solution value  = " << cplex.getObjValue() << endl;
        cplex.getValues(vals, var);
        env.out() << "Values = " << vals << endl;
    }
    catch (IloException& e) {
        cerr << "Concert exception caught: " << e << endl;
    }
    catch (...) {
        cerr << "Unknown exception caught" << endl;
    }
    env.end();
    return 0;
}  // END main

void read_graph(const int n, list <pair<int,int> > &adjacent_list)
{
  string line;
  int lineCnt=-1;
  for(int i=0;i<n+1;i++)
  {
    std::getline(cin, line);
    istringstream iss(line);
    int a;
    while (iss >> a)adjacent_list.push_back(make_pair(lineCnt,a));
    lineCnt++;
  }
}
```

Listing 4: csp_bipsolver_cplex.cpp

E CPLEX QUBO Solver for Chromatic Sum Problem

```
// qubo solver takes a matrix with size n and k as input and generates a
    optimal solution
// Usage:  $./qubo < matrix.txt
#include <ilcplex/ilocplex.h>
ILOSTLBEGIN
void read_qubo(int &n, double **&Q);
int main (int argc, char **argv)
{
    int n=0, k=0;
    double **Q;
    cin >> n >>k;
    read_qubo(n,Q);

    IloEnv    env;
    try {
        IloModel model(env);
        IloNumVarArray var(env);
        for (int i = 0; i < n; ++i) var.add(IloNumVar(env,0,1,ILOBOOL));
        IloExpr expr(env);
        for (int i=0; i<n; i++)
            for (int j=0; j< n; j++)
                expr += var[i]*var[j]*Q[i][j];
        IloObjective obj = IloMinimize(env,expr);
        model.add(obj);
        IloCplex cplex(model);
        cplex.setOut(env.getNullStream());

        // Optimize the problem and obtain solution.
        if ( !cplex.solve() ) {
            env.error() << "Failed to optimize LP" << endl;
            throw(-1);
        }

        IloNumArray vals(env);
        cplex.getValues(vals, var);
        // obtain sum of colorings
        int sl = 0;
        for (int i = 0; i < n/k; i++)
          for (int j = 0; j < k; j++) sl = sl + (int)vals[i*k+j] * (j+1);
        cout <<"('n=', "<<n <<", ' k=', "<<k<<")"<< endl;
        cout<<"newresult: [[";
        for (int i = 0; i < n; i++)
        {
          if(i+1!=n) cout <<(int)vals[i] <<", ";
          else cout <<(int)vals[i] <<"]]"<<endl;;
        }
    }
    catch (IloException& e) {
        cerr << "Concert exception caught: " << e << endl;
    }
    catch (...) {
        cerr << "Unknown exception caught" << endl;
    }

    env.end();
    for (int i=0; i<n; i++)
        delete [] Q[i];
    delete [] Q;
    return 0;
}  // END main

void read_qubo(int &n, double **&Q) {
  // Start of: initializing Q
```

```
    Q = new double*[n];
    for (int i=0; i<n; i++)
    {
        Q[i] = new double[n];
        for (int j=0; j<n; j++)
            Q[i][j] = 0;
    }
    for (int i=0; i<n; i++)
    {
      for(int j=0;j<n;j++)
            cin >> Q[i][j];
    }
}
```

Listing 5: csp_qubosolver_cplex.cpp