Nested dissection using quantum annealing

G.A.J. Custers

In cooperation with TNO

```python
class DomainWallGenerator(QUBOGenerator):
    def bqm_from_graph(self, G: nx.Graph) -> dimod.BinaryQuadraticModel:
        bqm = dimod.BinaryQuadraticModel("BINARY")
        for n in G.nodes:
            # Goal: minimize separator set.
            bqm.add_linear(2*n, self.params.seperator_penalty)

            # Constraint: domain-wall encoding constraint
            bqm.add_linear(2*n, 4*self.params.node_uniqueness)
            bqm.add_quadratic(2*n, 2*n + 1, -4*self.params.node_uniqueness)

        # Balance constraint
        bqm.add_linear_equality_constraint([x for n in G.nodes for x in (
            (2*n, 1),
            (2*n + 1, -2)
        )], self.params.get_balance_weight(len(G.nodes)), len(G.nodes))

        for (i, j) in G.edges:
            degree = G.degree[i]
            # Constraint: set A and B must not share edges.
            bqm.add_linear_from([
                (2*j + 1, int(self.params.independent_sets/degree)),
                (2*j, -int(self.params.independent_sets/degree)),
                (2*i + 1, int(self.params.independent_sets/degree)),
                (2*i, -int(self.params.independent_sets/degree))
            ])
            bqm.add_quadratic_from([
                (2*i + 1, 2*j + 1, -2*int(self.params.independent_sets/degree)),
                (2*i + 1, 2*j, int(self.params.independent_sets/degree)),
                (2*i, 2*j + 1, int(self.params.independent_sets/degree)),
            ])

            # Second goal: minimize edge cut
            bqm.add_linear_from([
                (2*i, self.params.edge_cut),
                (2*j, self.params.edge_cut)
            ])
            bqm.add_quadratic(2*i, 2*j, -2*self.params.edge_cut)
        return bqm

    def decode_solutions(self, G: nx.Graph, sampleset: dimod.SampleSet) -> Set[DissectionResult]:
```

Nested dissection using quantum annealing

by

G.A.J. Custers

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday April 17, 2025 at 10:30 AM.

Student number:	5119995	
Project duration:	January 8, 2024 – April 17, 2025	
Thesis committee:	Prof. dr. ir. R. E. Kooij ,	TU Delft, supervisor
	Dr. H. Wang,	TU Delft
	S. G. van der Linde,	TNO

TU Delft

Preface

Despite it being masked behind the cover of a scientific report, the journey of completing my Master's thesis has been one of the most difficult endeavors I've taken thus far. Unfortunately this was not due to the challenge of my research, but due to the state of my health. Therefore I would like to precede my acknowledgments by highlighting that many of these people did not just support me as an academic, but also as a human being.

Firstly, I'd like to thank the direct supervisors of my project, Rob Kooij, Stan van der Linde and Robert Wezeman. Without their knowledge, expertise and understanding this report would be not have seen the light of day. I'd also like to thank Milena Kooij-Janic, for overseeing my internship at TNO, and especially for the compassion she gave given my situation. I also want to thank my family, with special mention of Steven, for always managing to cheer me up, and being my quantum mechanics sparring buddy.

Lastly, I'd like to thank the countless of friends and acquaintances that have shaped my study period at the TU Delft. A special thank you to Pepijn and Niels, for being there no matter what, and knowing how to cheer me up even when the odds aren't looking amazing.

G.A.J. Custers
Rotterdam, April 2025

Contents

1 Introduction

1.1. Finite Element Analysis

In almost all science or engineering related field of study we find problems that are modeled by partial differential equations (PDE). These arise when the rate of change of a property is directly related to another one. For example, the heat equation models the heat diffusion through a medium, in which case the rate of change of heat over time is directly related to rate of change of heat over distance. In a few isolated cases, PDEs have known analytical solutions, however, most PDEs cannot be solved analytically. Instead, we often use numerical methods to provide an approximation to a solution. One of these methods is called Finite Element Analysis (FEA) or the Finite Element Method (FEM). As it turns out, this is a computationally intensive process, with the main computational load falling on that of solving systems of linear equations, which is commonly performed using Gaussian Elimination [28].

To illustrate, and to give context for the rest of this work, we present a small example. Suppose we have the following boundary value problem:

$$-\frac{\partial^2 u(x)}{\partial x^2} = f(x), \quad 0 < x < 1, \quad u(0) = 0, \quad u(1) = 0. \tag{1.1}$$

This formulation has strong requirements on the properties of $u(x)$. Namely, that it is twice differentiable. In many cases (such as on the interface of different materials or mediums) the first derivative is not continuous and as such the second derivative cannot be evaluated numerically. A naive approach could be to integrate both sides of (1.1) across its entire domain $0 < x < 1$, removing the second derivative.

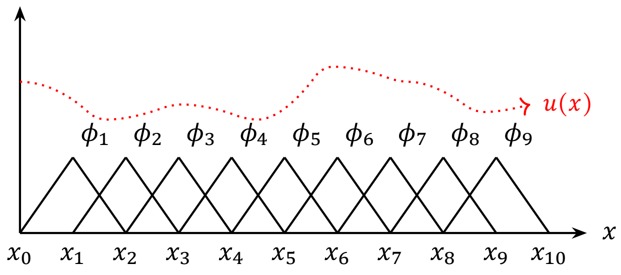
$$-\frac{\partial u}{\partial x} = \int_0^x f\,dx. \tag{1.2}$$

However, this integral requires the average of $f(x)$ and $-\frac{\partial u}{\partial x}$ to be equal across the entire domain. This is far from the original formulation, which required the functions to be equal at every point. Instead of integrating of the entire domain, we can integrate over a very small domain, splitting the original domain up into discrete elements.

We can concisely represent this by introducing a set of functions v called the test functions. One of such functions is non-zero only in a specific region of the domain, as such "sampling" the integral at different points. Multiplying (1.1) by v on both sides, and using integration by parts we arrive at the weak formulation. This procedure only works because we have set $u(0) = u(1) = 0$.

$$\int_0^1 \frac{\partial u}{\partial x}\frac{\partial v}{\partial x} dx = \int_0^1 f v\,dx. \tag{1.3}$$

To evaluate this expression we have to define v. We find v to be limited to a set of functions which are quadratic integrable ($\int_0^1 v^2 dx < \infty$) and $v(0) = v(1) = 0$ [45]. Now we can define our elements, which in the 1D case can be done simply defining points $x_i = ih$, where $i = 0, 1, ..., n$ and $h = \frac{1}{n}$, where n is the number of elements. On these elements we can define our test functions. There are many test functions we can use, often some piecewise linear function is used. For 1D this is the hat function,



$u(x)$
ϕ_1 ϕ_2 ϕ_3 ϕ_4 ϕ_5 ϕ_6 ϕ_7 ϕ_8 ϕ_9
x
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}

Figure 1.1: Imaginary solution $u(x)$ plotted with the discretized domain $x_0, \dots, x_{10}$ and the basis functions $\phi_1, \dots, \phi_9$.

as defined in (1.4). In higher dimensions this function has a similar shape, like a triangle in 2D, or a triangular pyramid in 3D.

$$\phi_i(x) = \begin{cases} \frac{x - x_{i-1}}{h} & \text{if} \quad x_{i-1} \leq x < x_i \\ \frac{x_{i+1} - x}{h} & \text{if} \quad x_i \leq x < x_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{1.4}$$

Now, we can express the solution as an approximation based on the test functions:

$$u_a(x) = \sum_{j=1}^{n-1} c_j \phi_j(x). \tag{1.5}$$

The constants c_j are unknowns that we need to find. To do so we first substitute the approximate solution $u_a(x)$ for $u(x)$ in the weak formulation in (1.3).

$$\sum_{j=1}^{n-1} c_j \int_0^1 \frac{\partial \phi_j(x)}{\partial x} \frac{\partial v}{\partial x}\, dx = \int_0^1 f v\, dx. \tag{1.6}$$

Recall that v is the set of test functions, so now to construct the system of equations we consider the equality in (1.6) for every $v \in \{\phi_1, \phi_2, \dots, \phi_{n-1}\}$.

$$\begin{bmatrix} \int_0^1 \frac{\partial \phi_1}{\partial x} \frac{\partial \phi_1}{\partial x}\, dx & \int_0^1 \frac{\partial \phi_1}{\partial x} \frac{\partial \phi_2}{\partial x}\, dx & \dots & \int_0^1 \frac{\partial \phi_1}{\partial x} \frac{\partial \phi_{n-1}}{\partial x}\, dx \\ \int_0^1 \frac{\partial \phi_2}{\partial x} \frac{\partial \phi_1}{\partial x}\, dx & \int_0^1 \frac{\partial \phi_2}{\partial x} \frac{\partial \phi_2}{\partial x}\, dx & \dots & \int_0^1 \frac{\partial \phi_2}{\partial x} \frac{\partial \phi_{n-1}}{\partial x}\, dx \\ \vdots & \vdots & & \vdots \\ \int_0^1 \frac{\partial \phi_{n-1}}{\partial x} \frac{\partial \phi_1}{\partial x}\, dx & \int_0^1 \frac{\partial \phi_{n-1}}{\partial x} \frac{\partial \phi_2}{\partial x}\, dx & \dots & \int_0^1 \frac{\partial \phi_{n-1}}{\partial x} \frac{\partial \phi_{n-1}}{\partial x}\, dx \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} \int_0^1 f \phi_1\, dx \\ \int_0^1 f \phi_2\, dx \\ \vdots \\ \int_0^1 f \phi_{n-1}\, dx \end{bmatrix} \tag{1.7}$$

The integral $\int_0^1 \frac{\partial \phi_i}{\partial x} \frac{\partial \phi_j}{\partial x}\, dx$ for a given i, j is related to the overlap of the functions ϕ_i and ϕ_j. Using Figure 1.1 we can see that most pairs of functions do not overlap. In fact, only neighboring functions have an overlap. Hence, most of the integrals in (1.7) evaluate to 0.

For the integrals on the diagonal we evaluate $\int_0^1 \frac{\partial \phi_i}{\partial x} \frac{\partial \phi_i}{\partial x}\, dx$. To do this we can use the derivative of ϕ_i:

$$\frac{\partial \phi_i}{\partial x} = \begin{cases} \frac{1}{h} & \text{if} \quad x_{i-1} \leq x < x_i \\ -\frac{1}{h} & \text{if} \quad x_i \leq x < x_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

and the fact that every interval (x_{i-1}, x_i) is of length h. We quickly see $\int_0^1 (\frac{\partial \phi}{\partial x})^2\, dx = \frac{2}{h}$.

As for the off-diagonal elements we can recognize that there is no overlap except when they are directly adjacent, meaning only $\int_0^1 \frac{\partial \phi_i}{\partial x} \frac{\partial \phi_{i+1}}{\partial x}\, dx$ is non-zero. In this case the integral evaluates to $-\frac{1}{h}$. Hence the full systems of equations is as follows.

$$\begin{bmatrix} \frac{2}{h} & -\frac{1}{h} & 0 & 0 & 0 & \cdots & 0 \\ -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & 0 & 0 & \cdots & 0 \\ 0 & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & 0 & \cdots & 0 \\ & & \ddots & \ddots & \ddots & & \\ 0 & 0 & 0 & 0 & -\frac{1}{h} & \frac{2}{h} & \frac{1}{h} \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{h} & \frac{2}{h} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ \cdots \\ c_{n-2} \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} \int_0^1 f\phi_1 \, dx \\ \int_0^1 f\phi_2 \, dx \\ \int_0^1 f\phi_3 \, dx \\ \int_0^1 f\phi_4 \, dx \\ \cdots \\ \int_0^1 f\phi_{n-2} \, dx \\ \int_0^1 f\phi_{n-1} \, dx \end{bmatrix} \tag{1.8}$$

Finally, to find our approximation for $u(x)$ we have to solve this system of equations. In the example Figure 1.1 we have quite a large discretization step, and hence the matrix size will be limited. However, large discretization also means large approximation error (we can calculate this error, but this is not relevant to the discussion in this work). As it turns out, in real world applications the matrices of the form shown in (1.8) can get extremely large. One method of efficiently solving these systems is to solve by using Gaussian elimination.

Gaussian elimination The matrix is equation shown in (1.8) can in general be expressed as $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x}$ is the vector of unknowns we wish to determine. An efficient way of solving this system is by transforming matrix $\mathbf{A}$ into the so called row-echelon form, which results in an upper triangular matrix $\mathbf{U}$. In $\mathbf{U}$ the last unknown is only related to a single value in $\mathbf{b}$, and so it can be found directly. Once we have determined the first unknown, we can substitute this into the following expression, which, due to the upper triangular matrix form, only depends on the last two unknowns. We can cascade this pattern until we have determined the entirety of $\mathbf{x}$ [15].

To transform the matrix into this row-echelon form, we can use Gaussian elimination, for which the process is as follows.

- Start at column $k = 0$ of matrix $\mathbf{A}$, we will make sure every entry in this column below the first row will be 0 (eliminated).
- For every row i below row k subtract $\frac{r_{i,k}}{r_{k,k}}\mathbf{R}_k$ from $\mathbf{R}_i$, where $\mathbf{R}_n$ is the n-th row of $\mathbf{A}$.
- Repeat this process for every column. Afterwards the matrix is in row-echelon form.

Pivoting It might be the case that the value $r_{k,k} = 0$, making the fraction $\frac{r_{i,k}}{r_{k,k}}$ undefined. To illustrate why this is an issue, we present the following elimination state.

$$\begin{bmatrix} \text{x} & \text{x} & \text{x} & \text{x} \\ & 0 & \text{x} & \text{x} \\ & \text{x} & \text{x} & \text{x} \\ & \text{x} & \text{x} & \text{x} \end{bmatrix} \tag{1.9}$$

Here, an x means a generic non-zero is present. Obviously if we terminated the elimination process when we encountered $r_{k,k} = 0$ unconditionally, we it would result in incorrect results. To mitigate this possibility, we can check if any of the rows $j < k$ have $r_{j,j} \neq 0$. We then swap the first j that satisfies this condition with row k, and proceed. This technique is called **pivoting**. In the context of this work we assume we do not require to perform pivoting, as considering the possibility of pivoting makes analysis much more difficult. FEM applications rarely require pivoting, so this assumption is acceptable for our desired application.

Fill-in The size and structure of the matrices resulting from FEM depend on how the elements are constructed. In many real world applications, there are many elements, on the order of millions, making the resulting matrices large [28]. Additionally, as is seen in the example (1.7), besides the diagonal, there is only a non-zero entry in the matrix when the elements overlap. In real world applications elements are not very densely connected, meaning that that many matrix elements will be zero, and do not contribute to subsequent computation. We refer to matrices like this as sparse matrices, where a majority of the elements are zero. This is in contrast to dense matrices, where most of the matrix entries are non-zero.

Sparse and dense matrices in general require different computational approaches, this is also the case for Gaussian elimination. Here the performance is highly dependent on the sparsity structure of the matrix. While the subtraction step in Gaussian elimination serves to eliminate elements in the matrix, in a sparse one it is not guaranteed that this step does not *add* elements to the matrix. Indeed the action will eliminate the entry in column k, but this action might also introduce entries where they were previously zero in any column below $i < k$. We can observe this is the case on the second row of the matrix in (1.8) already. Adding $\frac{1}{h}$ to the column will eliminate the leftmost $-\frac{1}{h}$, but introduce extra $\frac{1}{h}$ entries in place of zeros.

This phenomenon is called "fill-in", and is the center of much research in the numerical analysis domain. More specifically we can define fill-in as the introduction of non-zeros during the elimination process where there previously weren't any. Sparse matrices lend themselves to efficient storage by taking advantage of the fact that the zeros do not contribute to calculations. However, with decreased sparsity these storage formats and algorithms lose their efficiency. In practice the loss of efficiency due to fill-in is so severe that it is beneficial to perform pre-processing on the matrix to avoid the fill-in [28] [15]. We will elaborate on the analysis of fill-in in the following sections.

1.2. Graph theoretic model for fill-in

It is possible to represent $\mathbf{A}$ as a graph $G = (V, E)$, and analyze the elimination pattern from a graph theoretical perspective. A matrix can be represented as a graph by considering treating every matrix entry $a_{i,j}$ for a row i and column j as an edge (i, j) connecting node i to node j. The value of the matrix entry corresponds to the weight of the edge. If the graph is undirected, then the matrix is symmetric. In general we refer to the matrix representation of a graph as an adjacency matrix. Figure 1.2 shows an example matrix and its graph equivalent. Suppose in this example we wish to eliminate the first row. If we apply the first iteration of Gaussian elimination to the example matrix in Figure 1.2, the result is $\mathbf{A}_1$ in (1.10).

$$\mathbf{A}_1 = \begin{bmatrix} 1 & 3 & 2 & 5 & 2 \\ 0 & -8 & -6 & -15 & -6 \\ 0 & -6 & -3 & -10 & -4 \\ 0 & -15 & -10 & -24 & -10 \\ 0 & -6 & -4 & -10 & -3 \end{bmatrix} \tag{1.10}$$

Where $\mathbf{A}$ is sparse, $\mathbf{A}_1$ is far from it; most of the matrix is non- zero. This behavior is a direct result of the connectedness of the first row. The lack of non-zeros in the first row, and the presence of zeros in the other rows, gives rise to an increase in non-zeros after elimination. We can also model this using a graph visualization. Referring to Figure 1.2b we can see that node 1 (which is the first row of the matrix) is connected to every other node in the graph. The graph equivalent of fill-in is the addition of edges. In general the set of all edges added due to the elimination of a variable v is called the deficiency set, and is given by the expression in (1.11). The magnitude of this set is the fill-in.

$$Def(v) = \{(u, w) | (u, v) \in E, (v, w) \in E, (u, w) \notin E\}. \tag{1.11}$$

In simpler terms, (1.11) says that fill-in occurs whenever elimination of a node v causes nodes u, w that are connected through v to be disconnected. Then an edge (u, w) must be added to re-connect them. We can use this definition to define the elimination graph

$$G_v = (V - \{v\},\ E(V - \{v\} \cup Def(v)). \tag{1.12}$$

Assuming $n = |V|$, $Def_i(v)$ is the deficiency of v on a graph G_i, and an ordering on V is a bijection $\alpha : \{0, 1, ..., n\} \leftrightarrow V$ then we can calculate the all fill-in elements using

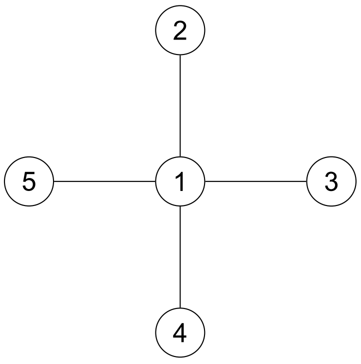
$$F(G, \alpha) = \bigcup_{i=1}^{n-1} Def_{i-1}(\alpha(i)). \tag{1.13}$$

The cardinality of $F(G, \alpha)$ is the total number of filled in elements for a given ordering α.

We can apply this definition of fill-in on the example in Figure 1.2b. Every vertex except itself is connected through vertex 1. Hence, eliminating vertex 1 disconnects every other vertex from each other, meaning $Def(1) = \{(2, 5), (5, 4), (5, 3), (2, 3), (2, 4), (3, 4)\}$, and $|Def(1)| = 6$.

$$\begin{bmatrix} 1 & 3 & 2 & 5 & 2 \\ 3 & 1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \\ 5 & 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) A matrix.



(b) The equivalent graph.

Figure 1.2: An example matrix and its equivalent graph representation.

1.3. The reordering problem

The addition of extra non-zeros into a sparse matrix can substantially increase the time to compute the Gaussian elimination. Extra working memory is required to store these non-zeros, which also increases the memory bandwidth requirement of the algorithm. Furthermore, sparse matrix formats are not usually designed with the possibility of updates in mind. This means that, depending on the data structures used, elimination of a row can become very expensive, as the data structure might need to be rebuilt. Lastly, it is more computationally expensive to perform computation on more values.

The cost of fill-in is in fact so high that there is a comprehensive academic selection of algorithms designed to reduce the fill-in associated with Gaussian elimination on sparse matrices [28] [15] [33] [23] [24]. To reduce the fill-in, we can change the order in which we perform elimination. In the previous section we've seen that the order with which we eliminate variables (or nodes) has an effect on the total fill-in we produce across the elimination process. It logically follows that we might be able to find an order of nodes that results in the lowest possible amount of fill-in. As it turns out, finding such an order for all possible matrix structures is NP-Hard [48].

As such, it's of interest to design heuristic algorithms that make a best effort to produce the lowest fill-in. In general, many proposed reordering algorithms use the abstraction of this elimination graph as a basis for their design. The cardinality of the deficiency set in the graph abstraction directly correlates to the expected fill-in in the matrix. Therefore, algorithms designed to find an order α such that $F(G, \alpha)$ is minimized, will also find a low matrix fill-in.

Reordering strategies broadly fall into two categories, those of the local pivotal kind, and those of the special form kind [15]. Local pivotal strategies are algorithms which consider the matrix only locally, and make a new decision on the ordering at every elimination step. An example of this is minimum degree ordering, where the next vertex to be eliminated is chosen to be the one with least degree [4]. Special form algorithms take a global approach, whereby the matrix is permuted into a "favorable" form [15]. These favorable forms give rise to augmented solving methods that result in less fill-in when compared with the original matrix. The exact workings of some common reordering strategies are covered in depth in Chapter. 2.

It is important to highlight that there is not a single strategy that is optimal for every use case. There are several factors that determine the performance of a reordering algorithm. Clearly the quality of the ordering (how much fill-in is produced in the end) is a factor. However, so is the computational requirement to get to this ordering. It is not beneficial to perform a preprocessing step if adding this step makes the entire solving process take longer. Another factor to consider is the structure and size of the input dataset. For example, minimum degree ordering generally works quite well on small datasets, while nested dissection works better for large datasets [15]. Finally, another important factor to consider is whether or not the algorithm lends itself to parallelization. There can be cases where a higher quality serial ordering has slower solving time than a parallelisable lower quality ordering.

This work focuses on nested dissection, a special form algorithm. This algorithm performs a particular form of graph partitioning, whereby the graph is split into two or more connected components. It does so by removing a set of vertices - hereafter referred to as the separator. In general the resulting ordering produces less fill-in when we are able to find a separator of minimal size. The algorithm is explained in more detail in Chapter 3. Finding a separator of minimal size is NP-Hard, which means state of the art implementations use a combination of heuristics to find a good separator [2] [31] [26] [27]. In this work we evaluate the use of a quantum computing powered heuristic for finding separators in the context of nested dissection.

1.4. Quantum computation

The field of quantum computing studies how we can leverage quantum physical effects for computation. At the heart of this innovation is the utilization of quantum bits, or qubits for short. In contrast to classical bits (which can only be of a discrete value of 0 or 1), qubits are subject to the principle of uncertainty, meaning that their value cannot be represented as a binary variable. Instead, we represent the value of a qubit as a linear combination of states, weighted by the probability of observing a particular state.

Bra-ket notation A single qubit system has two basis vectors. A common way of expressing these vectors is using Bra-ket notation. For the purpose of this work we can assume that $|v\rangle$ refers to a vector **b**, and $\langle v|$ refers to its complex conjugated transpose. Then to express the state of a single qubit system we write $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ and $|1\rangle = \begin{bmatrix}0\\1\end{bmatrix}$. α and β are complex numbers that represent the state of the system, such that $|\alpha|^2+|\beta|^2 = 1$. The values $|\alpha|^2$, $|\beta|^2$ can be interpreted as the probability of reading the respective state upon measurement. For example, the state $|\phi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ means that there is an equal probability of observing either $|0\rangle$ or $|1\rangle$. The phenomenon of probability dictating the measurement of a system is called superposition.

While superposition may appear counter productive for performing computation, this property can allow us to construct algorithms that have substantial speedup on problems that classical computers ordinarily struggle with. An example of such an algorithm is Grover's algorithm, which gives a quadratic speedup for unstructured search [36]. On a classical computer unstructured search takes $O(n)$, whereas Grover's algorithm can solve the same problem in $O(\sqrt{n})$.

Quantum annealing In the domain of quantum computers that are two main streams of computation: gate-based and annealing based. In gate based quantum computers algorithms take form as specialized quantum circuits, consisting of gates. These gates represent mathematical building blocks that operate on the state of qubits. In a gate based quantum algorithm calculations are performed on the probability of the qubit to collapse to a particular value. Probabilities can be adjusted such that performing a measurement has a high probability of reading a correct result for a given problem.

Quantum annealing (QA) is another paradigm of quantum computation. In contrast to gate based quantum computing, annealing focuses only on solving optimization problems of a particular form. Every problem submitted to a quantum annealer comes in the form of a quadratic unconstrained binary optimization (QUBO) problem [13]. This problem is then encoded in the physical quantum system. Such a system naturally tends towards its lowest energy state, i.e. the most favorable configuration given the present constraints. A quantum annealing system encodes the target given optimization problem in such a way that the lowest energy state of the quantum system corresponds to the most optimal solution of the QUBO problem. The specifics of quantum annealing are discussed in Chapter 4.

QA has been applied to several NP-Hard combinatorial problems, such as the graph coloring problem [30], graph partitioning problem [46] and even SAT solving [6].

Quantum operators In quantum mechanics the state of a quantum system is expressed using braket notation. The behavior of this system is modeled using the Schrödinger wave equation. The wave equation can be used to perform queries on various physical properties of the system. This is accomplished using Hermitian matrices named **operators**. An operator $\hat{A}$ can be applied to the state $|\psi\rangle$ of the system, if $|\psi\rangle$ is an eigenvector of the operator then

$$\hat{A}|\psi\rangle = a|\psi\rangle \tag{1.14}$$

, where a is the corresponding eigenvalue, representing the value of the queried property. There are many operators for the various physical properties of a quantum system. For example, the Hamiltonian operator $\hat{H}$ is one that queries the total energy of the system. The exact form of this matrix depends on the particular system it is querying. In this work we focus on the quantum Hamiltonians corresponding to the transverse field Ising model (TFIM), which are of the form

$$H = -\sum_{n=0}^{N-1} (g\sigma_n^x + \sigma_n^z \sigma_{n+1}^z) \tag{1.15}$$

[16] , where each σ_i is the magnetic spin of a particle along a particular axis. The TIFM is examined in more detail in Chapter 4.

1.5. Goals

In general graph partitioning is a difficult optimization problem to solve. There are many existing heuristics (some of which are examined in Chapter 3), but with the new development of quantum annealing as a meta-heuristic, it is interesting to see how well this method can be applied to graph partitioning in the context of numerical analysis.

To this end, this work aims to answer the following research questions:

- How can we formulate the nested dissection graph partitioning problem as an optimization problem suited for quantum annealing?
- How well does this formulation perform when compared to other heuristic methods for performing nested dissection?
- What does the inclusion of a quantum step mean for the Finite Element Analysis pipeline?

2 Related works

The question of matrix reordering is one that has seen extensive research for many years. In general, the community has settled on two main reordering strategies, local pivotal and special form methods [15]. While both methods are heuristics, local pivotal methods are of the greedy kind and special form methods use divide and conquer strategies. The main advantage of special form methods is not necessarily in the reduction in fill-in, but rather that they are better suited for parallelised processors. This is in contrast to the local pivotal methods, which are by definition difficult to parallelise.

In this section we examine several methods to reorder matrices, and other quantum linear solving methods.

2.1. Local pivotal reordering methods

Local pivotal methods work in close tandem with the Gaussian elimination process. At every stage k the reordering method determines the next pivot $a_{ij}^{(k)}$ in the $(n-k)\cdot(n-k)$ submatrix that minimizes some expression. This expression differs per local pivotal method, and several criteria will be explored in this subsection.

2.1.1. Markowitz method

One of the earliest ordering strategies was proposed by Markowitz in 1957 [37]. For each row i in the submatrix, r_i^k denotes the number of non-zero entries in said row, at Gaussian step k. Similarly, for each column j, c_j^k denotes the number of non-zero entries in said row. Then the Markowitz criterion is

$$(r_i^k - 1)\cdot(c_j^k - 1), \tag{2.1}$$

which is minimized at each step of the elimination process. This expression can be interpreted as finding the pivot for each iteration which modifies the least coefficients for the remaining submatrix. Implementation of this criterion is not trivial, as it requires knowledge of the sparsity structure of every k-th submatrix. This is apparent when considering that minimizing (2.1) means knowing r_i and c_i for every i and j.

2.1.2. Minimum degree ordering

When we know our matrix is symmetric, the above criterion simplifies. More specifically, it is now enough to find an i such that $r_i^{(k)}$ is minimized, and using $a_{ii}^{(k)}$ as pivot. Since the matrix is symmetric, the rows and columns have the same sparsity pattern, meaning we only need information on one of the axes. This technique is called minimum degree ordering, and is a very popular method. Finite element methods often produce symmetric matrices.

The name minimum degree ordering comes from the graph interpretation of the matrix. This interpretation comes naturally when the matrix is symmetric. Minimizing $r_i^{(k)}$ in this case is equivalent to choosing the vertex i that has the lowest degree in the graph. Despite being a conceptually simple algorithm, lots of research has been done to improve the computation time. While only having to scan all rows on every iteration, on large matrices a naive implementation might not be performant enough to justify re-ordering the matrix. Below a few innovations on the minimum degree algorithm are summarized.

Multiple minimum degree The multiple minimum degree algorithm, proposed by Liu et al [35] uses several observations to reduce the storage of and work done on the elimination graph. To support the computation an alternative data structure is used instead of the elimination graph to simulate the elimination process. The datastructure used is called a quotient graph, which primarily stores cliques instead of nodes. In a clique it is not required to track edge information, just a list of nodes suffices. Therefore storage requirements of these graph are shown to be no worse than the elimination graph, and are often better.

One of the optimizations identified in multiple minimum degree is that of indistinguishable nodes. The adjacency set $\text{Adj}_G(i)$ of a node i on graph G is the set of all nodes connected to i. Suppose nodes i and j satisfy $\text{Adj}_G(i) \cup \{i\} = \text{Adj}_G(j) \cup \{j\}$, then they can be eliminated in any order while maintaining the same fill-in; in other words, they are indistinguishable. This observation can be used to delay the degree computation. Instead of recomputing the degree in the neighborhood of elimination for each eliminated node, one can recompute the degree only when the entire set of indistinguishable nodes are eliminated.

We can use the concept of indistinguishable variables to identify “supernodes”. These are sets of indistinguishable nodes. It is possible to also order by the degree of the supernodes instead of their constituent nodes. This is called the external degree, defined as $d_i = t_i - |\mathbf{i}| + 1$, where t_i is the degree of some node $i \in \mathbf{i}$, called true degree. It has been shown that ordering by external degree instead of true degree results in better orderings.

Another way to delay degree computation is using the concept of outmatched nodes. A node i is said to be outmatched by a node j if $\text{Adj}_G(i) \subseteq \text{Adj}_G(j)$. This also implies that degree$(i) \leq$ degree(j). Therefore, we don't need to update the degree of j until i has been eliminated.

Finally, we can further delay degree computation using multiple elimination. Instead of considering only a single pivot per elimination round, we consider multiple independent pivots of the same minimum degree. More specifically, eliminating i only changes nodes *not* in $\text{Adj}_G(i)$. Therefore, we can look for a new pivot in the subgraph $G - (\text{Adj}_G(i) \cup \{i\})$. If this pivot has the same degree as i then we can eliminate it in the same step. This process is continued until the subgraph is empty.

Approximate degree Approximate degree is an algorithm that builds on multiple minimum degree, using an optimized method of computing the external degree of nodes [1]. Instead of computing an exact value of d_i, it computes an upper bound $\bar{d}_i$. In the k-th elimination step node p is chosen as pivot, $\bar{d}_i^k$ is given by

$$\bar{d}_i^k = \min \begin{cases} n - k, \\ \bar{d}_i^{k-1} + |\mathcal{A}_p \backslash \mathbf{i}| \\ |\mathcal{A}_i \backslash \mathbf{i}| + |\mathcal{A}_p \backslash \mathbf{i}| + \sum_{e \in \mathcal{E}_i \backslash \{p\}} |\mathcal{A}_e \backslash \mathcal{A}_p| \end{cases} \tag{2.2}$$

, where $\mathcal{A}_i$ is the set of non-eliminated nodes adjacent to i and $\mathcal{E}_i$ is the set of eliminated variables adjacent to i. It turns out that this upper bound is much easier to calculate then the exact external degree, resulting in a substantial asymptotic speedup.

2.1.3. Minimal fill-in

We can also consider another greedy ordering criterion instead of the Markowitz criterion. One method is to use a local minimal fill-in criterion. At every elimination point we search for the pivot which will cause the least amount of fill-in. Like previous local pivotal strategies, this approach is not a global one, so it is not guaranteed to find a globally optimal ordering.

When compared to other local pivotal strategies, local minimal fill-in has higher computational costs. This is because one not only needs to know the sparsity pattern at every step, but must also calculate the amount of fill-in each node generates. It has not been shown that minimal fill-in produces better orderings than other local pivotal strategies. Combined with the higher computational costs, this strategy is not popular.

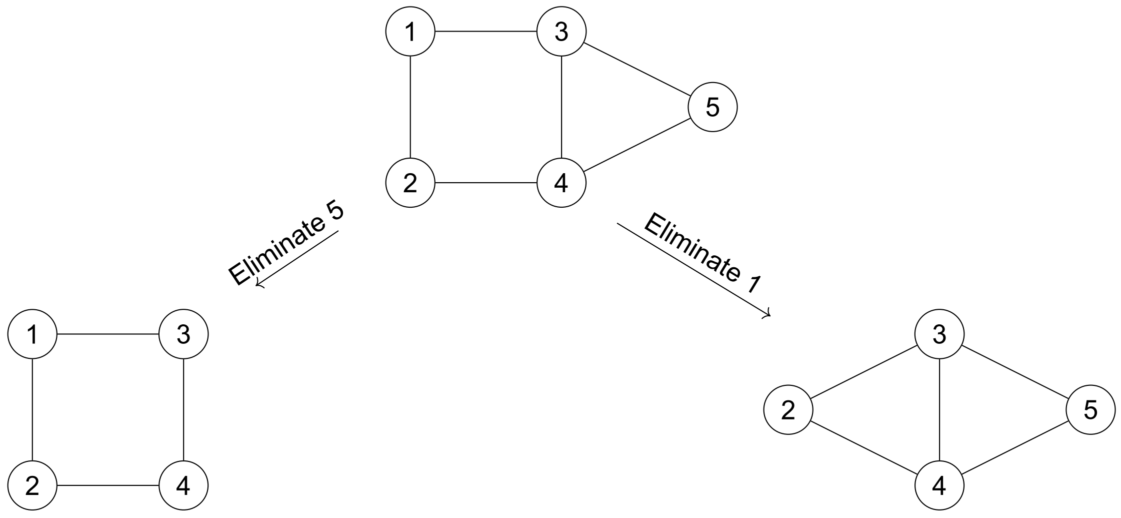


Figure 2.1: Example of a graph in which a tie-breaking choice leads to a difference in fill-in.

2.1.4. Tie breaking

Local pivotal methods are simple and effective algorithms for reducing fill-in, but unfortunately their performance characteristics are difficult to determine. There are some graph classes for which performance bounds are known. For example, it is known that minimum degree is optimal for tree graphs [15]. However, the amount of problem classes for which the performance of local pivotal methods is known is very limited.

The primary reason for this is the problem of tie-breaking. To illustrate tie-breaking, suppose we apply minimum degree ordering on the graph shown in Figure 1.2. In this figure we start with a simple graph at the top. Here vertex 1 and 5 both have a degree of two. Therefore there are two valid choices for which node to eliminate first under minimum degree ordering. In this small example the subsequent graphs show that this choice indeed has an effect on the fill-in of the ordering. If we begin by eliminating vertex 5 we see that the resulting graph is a 2x2 grid. Fully eliminating the 2x2 grid results in single filled in edge. However, if we instead choose to eliminate vertex 1 first, then we obtain a graph which has no fill-in when fully eliminated. In the presented example the overall fill-in is equal, irrespective of the initial choice, however, it has been demonstrated in numerical experiments that the tie-breaking choice has significant effect on the total fill-in [24].

The literature on tie-breaking strategies for minimum degree ordering is sparse. The SuperLU linear solver package mentions the ordering strategies used, but makes no mention of the tie-breaking strategy it uses for minimum degree [32]. George et al. reported in 1989 that the popular minimum degree implementations all relied on a randomized tie breaking strategy [24].

2.2. Special form methods

We have discussed ordering methods that locally minimize some factor such as degree or fill-in. In this section we consider algorithms that don't directly minimize a specific criterion, but instead aim to permute the matrix into some particular form. For nested dissection the matrix is transformed into an arrow-head matrix. In Cuthill-McKee [11] the resulting matrix is a band matrix.

In a band matrix there is no fill-in outside of the band, so the off-diagonal component remains untouched, and does not need to be stored. Minimizing the bandwidth of the band matrix reduces the fill-in. The bandwidth of a matrix is defined as $2m + 1$, where the m is the semibandwidth, which is in turn defined as the smallest integer such that when $a_{ij} = 0$, $|i - j| > m$. Cuthill-McKee aims to find a restructuring of the matrix resulting in lowest bandwidth.

Nested dissection creates a different matrix structure, namely an arrow-head matrix. This is a blocked matrix with a diagonal component, and a filled border in the last row and column. Since this algorithm is extensively examined in Chapter 3, we defer to that chapter for more information on nested dissection.

2.2.1. Cuthill-McKee

The Cuthill-McKee algorithm makes use of so called level sets [11]. These are sets of nodes constructed from the neighbors of nodes in a lower level set. To construct a level set one can use the following procedure.

- Take an initial set S_1 consisting of a single node, the starting node.
- Every other set S_i, $i \neq 1$ is filled with the neighbors of all the nodes in S_{i-1}.

Cuthill-McKee builds upon this principle, and also provides a method to order the nodes within each S_i. During construction of S_i, the neighbors of the vertex ordered the previous level S_{i-1} are explicitly ordered first, and subsequently the neighbors of the vertex ordered second in the level S_{i-1} are ordered second, etc. The result of following this procedure to relabel a matrix is a band matrix, with each block in the band corresponding to a level set S_i, with a smaller bandwidth. The bandwidth is directly related to the size (and therefore number) of the level sets. Smaller level sets (and consequently more of them) results in a lower bandwidth. It turns out that reversing the Cuthill-McKee ordering lowers the produced fill-in [15]. Hence the algorithm is often referred to as Reverse Cuthill-McKee (RCM).

Obviously the starting node will have an effect on the magnitude of the bandwidth. It is not immediately apparent which starting node is going to result in the lowest bandwidth. There are many proposed heuristics for choosing the starting node, but one popular one is a greedy approach. In this approach we try to optimize the number of level sets by trying each node in the last level set S_k as a new starting point. We continue this process until a new starting node does not result in higher k. The initial starting point is chosen at random.

2.3. Existing quantum methods

Aside from quantum annealing there are also multiple other quantum methods available for numerical analysis. Some of these approaches aim to completely replace Gaussian elimination, while others put focus on the reordering problem. A few of the most relevant quantum methods are summarized in this section.

2.3.1. HHL

HHL [14] is a gate based quantum algorithm for solving the Quantum Linear System Problem (QLSP). QLSP is the quantum analogue of solving $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{x}$. Formally, the QLSP is:

$$|x\rangle = \frac{\mathbf{A^{-1}}\,|b\rangle}{|\mathbf{A^{-1}}\,|b\rangle\,|} \tag{2.3}$$

The approach HHL takes to solve the QLSP is a spectral one. One can get an intuition for the approach HHL takes by considering we can express a matrix $\mathbf{A}$ using the outer product of its eigenvectors $\mathbf{u}_i$.

$$\mathbf{A} = \sum_{j=0}^{N-1} \lambda_j \,|\mu_j\rangle\,\langle\mu_j| \tag{2.4}$$

, where λ_i is the eigenvalue corresponding to the i-th eigenvector $\mathbf{u}_i$.

Using the fact that an eigenvalue λ_i for $\mathbf{A}$ means $\mathbf{A}^{-1}$ has an eigenvalue λ_i^{-1}, we can express $\mathbf{A}^{-1}$ in a similar way.

$$\mathbf{A}^{-1} = \sum_{j=0}^{N-1} \lambda_j^{-1} \,|\mu_j\rangle\,\langle\mu_j| \tag{2.5}$$

We can rewrite $\mathbf{b}$ as a linear combination of $\mathbf{A}$'s eigenvectors.

$$\mathbf{b} = \sum_{j=0}^{N-1} \beta_j \,|\mu_j\rangle \tag{2.6}$$

Then $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ can be expressed in terms of $\mathbf{A}$'s eigendecomposition.

$$\mathbf{x} = \sum_{j=0}^{N-1} \lambda_j^{-1}\beta_j \left|\mu_j\right\rangle \tag{2.7}$$

Thus, assuming we can efficiently compute the eigenvalues of $\mathbf{A}$, we can easily calculate $\mathbf{x}$. This procedure has running time of $O(\text{poly}\log N)$. HHL is able to accomplish this by combining various known quantum buildings blocks, primarily the Quantum Phase Estimation (QPE), a quantum algorithm that finds eigenvalues. A detailed explanation of the HHL procedure is out of the scope of this work. Nevertheless, a high level overview of the workings is presented.

1. Input vector $\mathbf{b}$ is encoded in qubit form.
2. Use Hamiltonian Simulation to apply $e^{i\mathbf{A}t}$ to $|b\rangle$ over a superposition of t.
3. Use QPE to find the eigenvalues of $e^{i\mathbf{A}t}$ and decompose $|b\rangle$ into the eigenbase of $e^{i\mathbf{A}t}$.
4. Use controlled rotations to calculate the inverse eigenvalues.
5. Extract the calculated values out of the quantum system.

Limitations While the HHL algorithm is a valuable tool, with many different theoretical applications [14], such as for example electrical network analysis [47], it does not come without limitations.

Firstly, due to the fact that both the input and output are encoded as a quantum system, input and output is not straightforward. Both reading and writing has time complexity $O(n)$, suppressing the asymptotic speedup. This places limitations on the application HHL should be used in. For the input, HHL performs well in situations where the input is generated by some other quantum process. Moreover, HHL is only useful if one is not necessarily interested in the exact value of $\mathbf{x}$, but rather the expectation value of $\langle x|\mathbf{M}|x\rangle$, where $\mathbf{M}$ is some Hermitian matrix.

Secondly, the nature of the Hamiltonian Simulation algorithm places limitations on the sparsity structure of $\mathbf{A}$. Hamiltonian Simulation is a quantum algorithm that approximates the evolution operator $e^{i\mathbf{A}t}$. The complexity bound for Hamiltonian Simulation is $O(\log n \frac{s^4\kappa^2}{\epsilon})$, where s is an upper bound on the number of non-zeros for each row of $\mathbf{A}$, κ is the condition number for $\mathbf{Ax} = \mathbf{b}$ and ϵ is an error term. Importantly, s and κ affect the exponential speedup that Hamiltonian Simulation provides. This places a limitation on the types of systems that can be efficiently solved with HHL. One type of system that typically has matrices with high condition number is in fact a finite element system [18]. This makes HHL unsuited to finite element applications.

2.3.2. Variational solvers

In reality the gate depth of HHL instances quickly outgrow the current physical possibilities. This is not limited to just HHL, and many other gate based quantum algorithms do not outperform their classical counterparts due to technological limits. Although current quantum computers are not at the necessary scale for running these complex algorithms, they *can* be used for running hybrid algorithms. These are algorithms whereby a combination of classical and quantum techniques is utilized to produce the desired result. One of such algorithms is called the Variational Quantum Eigensolver (VQE).

VQE VQE is able to efficiently determine the expectation of some observable over some wavefunctions using quantum methods [20]. Computing these expectations on classical platforms is exceedingly difficult. The input state for which the expectation is determined is varied by a classical component. In this way, a classical optimizer can vary the input of the system, and use the quantum observable as a heuristic. In many cases the input is varied such that the observable is minimized.

VQLS VQE has many applications, this includes solving linear systems. VQLS is an adaptation of VQE applied to the QLSP [7]. In practice ”adapting” means defining an ansatz for the gate sequence, and a cost function which the classical part optimizes. To solve a linear system, the algorithm takes as input some matrix $\mathbf{A}$ and some vector $\mathbf{b}$. VQLS assumes that $\mathbf{A}$ is represented as a linear combination of unitary matrices, and that this representation can be efficiently represented as a set of quantum gates. One can use a quantum algorithm for expressing a sparse matrix as unitaries, the complexity of which depends on, among other factors, the sparsity of the matrix [5]. This is a similar assumption to the one made in HHL, where an efficient preparation of input state is necessary for strong performance. The input vector $\mathbf{b}$ is taken as a gate sequence U, producing a quantum state $|b\rangle$, which is proportional to $\mathbf{b}$.

Once the inputs are determined, we can start the optimization process. During the optimization phase the inputs, along with a gate sequence $V(\alpha)$ are used to produce a value for the cost function $C(\alpha)$. A classical computer then uses some optimization algorithm to solve $\min_{\alpha} C(\alpha)$. Recently it has become popular to use gradient descent as optimization method. This can be done efficiently by recognizing that $C(\alpha)$ is differentiable, and thus the gradient of the cost function can be directly computed using the quantum circuit. The final value for α is used in the quantum circuit $|x(\alpha)\rangle = V(\alpha)|0\rangle$. For an optimal α_{min}, $|x(\alpha)\rangle$ is proportional to the solution $\mathbf{x}$.

2.3.3. Quantum minimum fill

The previous two quantum algorithms for solving linear systems have been focused around gate based systems. However, as outlined in the introduction, gate based approaches are not the only quantum approaches one can take. Indeed, there is also prior work on the topic of using quantum annealing to aid linear solvers. In particular, Komiyama et al. present a combinatorial formulation of the minimum-fill problem, which they call the Quantum Minimum Fill algorithm [29].

To model the minimum-fill problem as a QUBO problem the authors construct a matrix and vector of binary values. A matrix entry a_{ij} represents a node i being removed at elimination step j. The vector entry v_i is 1 when edge i needs to be added to the graph during the elimination process. Each pair of nodes in the graph that does not already have an edge between them receives an entry in this vector. The model's optimization goal is to minimize the number of 1's present in the edge vector, i.e. $\min \sum_{e \in E} e$. To ensure edges get appropriately added, several constraint expressions are used. Firstly, the model assumes that there is only one node eliminated in each step, which is enforced by $\sum_i (\sum_j a_{ij} - 1)^2 = 0$. Secondly, a constraint must be introduced that enforces the addition of edges into the edge vector. An edge is added when:

- A node u is to be deleted
- And nodes i and j in the neighborhood of u have not already been deleted.
- And node i and j both have an edge connected to u.

These constraints are enough to simulate the process of elimination in QUBO form. A minimization of the QMF Hamiltonian yields the order with the lowest possible fill-in for the given problem.

Limitations While Quantum Minimum Fill provides an algorithm to solve the minimum fill problem, it might be difficult to realize practically for appreciable problem sizes. This stems from the poor scalability in terms of required qubits. For solving an nxn matrix the algorithm needs at least n^2 binary variables. Crucially, this is the amount of binary variables required, and not qubits. In reality, due to topological limitations, the amount of actual qubits required on the quantum annealer will be considerably larger than the amount of binary variables. This is compounded by the fact that this QUBO formulation has a large number of connections between its variables. Physical quantum computers do not support all-to-all connectivity on their qubits, the number of couplers is limited. This means that for a physical realization of the QUBO problem, the number of qubits exceed the number of binary variables, due to the effects of embedding.

3
Nested Dissection

Nested dissection (ND) is a heuristic divide and conquer algorithm for solving the minimum fill-in problem. The algorithm aims to produce an elimination order that results in an arrow-head matrix with minimal bandwidth. The arrow-head matrix is a type of square matrix where the non-zeros only lie on the diagonal and on the last row and column. ND was first introduced by George et al. in 1973 [23], applying it on grid graphs as obtained in finite element methods. They showed that on this graph class nested dissection is able to find the asymptotic optimal ordering.

Section outline This section begins by explaining the nested dissection algorithm in detail, and the intuition for why it is a successful heuristic. We examine why making a nested dissection ordering is difficult in the general case. Some classes of graphs have a known bounded quality on nested dissection, these are outlined. We provide an overview of the state of the art ND implementation MeTiS that shall be used as a benchmark for classical methods.

3.1. The algorithm

Nested Dissection is a special form algorithm, meaning that it aims to globally restructure a given matrix $\mathbf{A}$ into a specific form. Particularly, ND tries to re-order the matrix into an arrow-head form where the off-diagonal components are as large as possible. An arrow-head matrix is shown in (3.3). There exists an alternate direct solving method for this form of matrix, whereby the diagonal blocks undergo elimination, but the off-diagonal blocks stay intact. In this way only the diagonal blocks experience fill-in, while the off-diagonal blocks keep their sparsity [15]. A lower bandwidth of the diagonal is therefore also associated with a lower fill-in. An optimal nested dissection ordering is one that produces the lowest possible bandwidth in the diagonal.

Graph dissection To produce this arrow-head matrix structure we relabel nodes in a graph such that neighboring nodes are ordered close together in the matrix. Take for example the matrix structure corresponding to the graph shown in Figure 3.1:

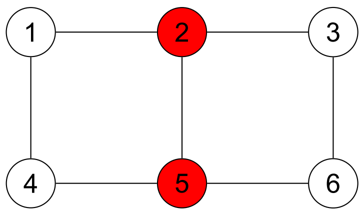


Figure 3.1: Example of a dissection, removing the red nodes from the graph and their incident edges results in two distinct connected components.

$$\begin{bmatrix} \text{x} & \text{x} & & \text{x} & & \\ \text{x} & \text{x} & \text{x} & & \text{x} & \\ & \text{x} & \text{x} & & & \text{x} \\ \text{x} & & & \text{x} & \text{x} & \\ & \text{x} & & & \text{x} & \text{x} \\ & & \text{x} & & \text{x} & \text{x} \end{bmatrix} \tag{3.1}$$

The goal is to relabel the rows and columns of this matrix such that most of the non-zero entries are concentrated around the diagonal. One way of achieving this is by subtracting a set of vertices and their associated edges such that the graph is split into two separate connected components. This is called graph dissection. In the case of our example a dissection is shown in Figure 3.1 by way of the red highlighted vertices 2 and 5. It is clear that when the red vertices are removed the graph is split in half, and there are two equally sized connected components $A = \{1, 4\}$ and $B = \{3, 6\}$ left. We can take the red vertices and place them in a set $S = \{2, 5\}$. This is named the separator set.

We can then observe what happens when we order the vertices in A first, then the vertices in B and the vertices in S. Reordering here means that the rows *and* columns are interchanged to match the new ordering. This changes the structure of the matrix.

$$\begin{bmatrix} \text{x} & \text{x} & & & \text{x} & \\ \text{x} & \text{x} & & & & \text{x} \\ & & \text{x} & \text{x} & \text{x} & \\ & & \text{x} & \text{x} & & \text{x} \\ \text{x} & & \text{x} & & \text{x} & \text{x} \\ & \text{x} & & \text{x} & \text{x} & \text{x} \end{bmatrix} \tag{3.2}$$

The sparsity structure of the matrix in (3.2) now resembles that of (3.3), where each $\mathbf{A}_{i,i}$ is a 2x2 matrix. In this form the occurrence of fill-in is constrained to the diagonal and border blocks. No fill-in occurs in the empty off-diagonal matrices. Herein lies the core principle behind dissection based heuristics. The heuristic aims to maximize the size of the empty off-diagonal blocks, thereby localizing the fill-in as much as possible. The size of the separator set determines the size of the off-diagonal blocks, the smaller the separator, the larger the off-diagonal component.

Computing a separator Finding the smallest possible separator turns out to be a computationally difficult problem. This problem is called the vertex separator problem (VSP), and has been proven to be NP-Hard [3]. The VSP on a graph $G = (V, E)$, with $n = |V|$ and a $\beta(n) \leq n$, is to find sets A, B, S such that

1. there is no $(i, j) \in E$ such that $i \in A$ and $j \in B$,
2. $\max\{|A|, |B|\} \leq \beta(n)$,
3. $|S|$ is as small as possible.

We say a separator is valid when it satisfies both constraint 1 and 2. It is optimal when it is valid and satisfies 3.

The algorithmic complexity of this problem depends on the definition of $\beta(n)$. There are some cases where a polynomial solution is known. An example of this is the k-connectivity problem, which is colloquially defined as finding the smallest k such that the graph becomes disconnected when k vertices are removed. We can express this in terms of the VSP by using $\beta(n) \leq n - k - 1$. A popular algorithm to calculate the k-connectivity uses a maximum flow procedure [19]. An augmented graph is constructed, where each vertex v in the graph is split into a v_{in} and v_{out}, with an edge of capacity one to connect them. Every edge $(u, v) \in E$ on the original graph becomes (u_{out}, v_{in}) on the augmented graph. Finding the paths that contribute to the maximum flow between a vertex pair (a, b) on the augmented graph is equivalent to finding all paths from a to b with a unique vertex set. A path with a unique vertex set is referred to as a disjoint vertex set. The number of disjoint vertex sets is equal to amount of vertices needed to disconnect a from b. To find the k-connectivity we iterate over all pairs $(a, b),\ a \neq b$ in the graph and find the lowest number of disjoint vertex sets. In the worst case this procedure has a running time of $O(n^5)$.

In the case of graph dissection, we are interested in forms of the VSP where $\beta(n) = \alpha n$, for some constant α. This is a balanced VSP, and is fundamentally different from the k-connectivity problem described above. The k-connectivity problem considers the graph locally, determining the partition using properties of pairs of nodes. However, solving a balanced VSP requires a global consideration of the graph, which is significantly more difficult. There are some combinations of α and graph classes that are known to have polynomial solutions. We say a graph satisfies a $f(n)$-separator theorem when it is known that a separator can be found that satisfies $|S| \leq \beta f(n), \quad \beta > 0$. Grid graphs, like the ones shown in Figure 3.1 are known to satisfy a $\sqrt{n}$-separator theorem [34]. This result is used in the original paper describing nested dissection on grid graphs [23]. Planar graphs are found to have a $\sqrt{n}$-separator theorem with $\beta = 2\sqrt{2}$ and $\alpha = \frac{2}{3}$. This separator can be found in $O(n)$ [34]. Graphs of bounded genus also satisfy the $\sqrt{n}$-separator theorem, with $\beta = \sqrt{g}$, where g is the genus of the graph [33]. Chordal graphs satisfy a similar condition, they satisfy a $\sqrt{m}$-separator theorem, with m being the number of edges in the graph [25].

Nested dissection Nested dissection is a recursive application of graph dissection [23]. Every graph dissection produces two subgraphs that can be dissected another time. This procedure differs from a single dissection in the sense that the submatrices A and B are also structured as an arrow-head matrix [15]. On large graphs it is often the case that dissecting the subgraphs improves the matrix structure. The following is the complete procedure for nested dissection on a graph $G = (V, E)$, derived from adjacency matrix $\mathbf{A}$.

- Find a vertex set $S \subset V$ such that the removal of those vertices split G into two independent vertex sets A, B. Sets A, B make two subgraphs that share no edges between them.
- Each subgraph S, A, B has a corresponding submatrix. Rearrange $\mathbf{A}$ using these submatrices, placing each submatrix on the diagonal, A first, B second and S last.
- Recursively apply these steps to A and B, until some stopping condition is met.

Completing this process rearranges matrix $\mathbf{A}$ into an arrowhead like structure.

$$\mathbf{A_r} = \begin{bmatrix} \mathbf{A_{11}} & & \mathbf{A_{13}} \\ & \mathbf{A_{22}} & \mathbf{A_{23}} \\ \mathbf{A_{31}} & \mathbf{A_{32}} & \mathbf{A_{33}} \end{bmatrix} \tag{3.3}$$

The sub-matrices $\mathbf{A}_{i,j}$ are not of uniform size. The off-diagonal sub-matrices are the connections between the found subgraphs.

Due to the recursive nature of the nested dissection procedure, it lends itself well to parallelization. Every submatrix can be dissected in parallel. Moreover, because a large chunk of the matrix is independent of each other, one can also eliminate those submatrices in parallel. This is one of the largest advantages of nested dissection. While some other heuristics like MDO might be able to order with lower fill-in, they are generally not parallelisable. For very large matrices, where computation without vector processors is not feasible, this is an important feature.

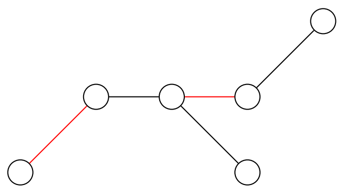
3.2. MeTiS

While there are some known polynomial solutions to the VSP, these are valid for specific graph types. The matrices resulting from FEM are not necessarily one corresponding to graph type satisfying a $\sqrt{n}$-separator theorem. On these matrices we must use heuristics to compute the dissection. MeTiS [27] is a state of the art package that uses a multilevel heuristic to compute the dissection. It implements a nested dissection routine, along with some other local pivotal methods which are used when the graph size is small. In this section we examine the MeTiS implementation in detail.

3.2.1. Multilevel dissection

A multilevel dissection is a dissection strategy whereby the partitioning is performed on a much smaller graph obtained through a graph coarsening method. The high level algorithmic overview is as follows.

- Reduce the input graph size using graph coarsening.


(a) A maximal, but not maximum matching.

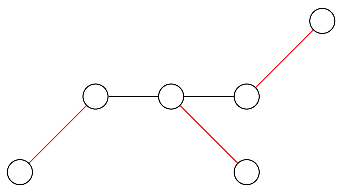
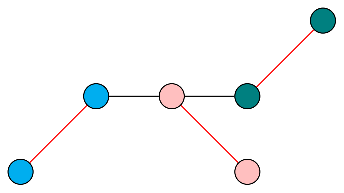

(b) A maximal and maximum matching.

Figure 3.2: An example graph showing two possible maximal matchings, one of which is maximum. The edges marked in red are part of the matching set.


(a) The non-coarse graph.

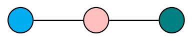

(b) The corresponding coarsened graph.

Figure 3.3: An example showing how to use a matching to coarsen the graph. Vertices are merged together when they are part of the same matching. The merged vertices are marked in the same color.

- Perform partitioning on coarsened graph.
- Expand graph to original size, attempting to improve the partition at every expansion step.

The benefit of multilevel dissection is that it allows usage of more complex partitioning algorithms. As with any heuristic solving NP-Hard or Complete problems, there is a trade-off between quality and speed. Various graph partitioning algorithms are discussed in Section 3.2.3. In short, there are some graph partitioning schemes that take a prohibitively long time to execute on large problem sizes, but also result in considerably higher quality partitions. Therefore, it is worthwhile to use graph coarsening to scale the graph down to a size where it becomes viable to use these more expensive partitioning methods.

In the rest of this section we expand on the various available coarsening and partitioning strategies.

3.2.2. Graph coarsening

The general goal of graph coarsening is to transform a large graph into a structurally similar smaller graph. MeTiS aims to coarsen a graph until it is below 100 vertices. A fundamental part of their algorithm makes use of graph matching. To understand how MeTiS coarsens graphs, we must first understand the process of matching.

Maximal matching A matching in the context of graph theory is a set of edges that are non-adjacent. In other words, a set of edges for which each of the edges do not share a vertex with each other. A maximal matching is a matching in which every edge of the graph is connected to at least one matched vertex. In contrast, a maximum matching is one where the number of matched edges is maximized. Figure 3.2 shows an example graph with two possible maximal matchings, one of which is a maximum matching.

We can use a matching to produce a coarser, but structurally similar version of the graph. Every pair of nodes u, v belonging to the same matched edge (u, v) is merged into a single node. In this process the weight of each pair of nodes is summed to produce the weight of the new merged node w. We then use $E_w = E_u \cup E_v \setminus \{(u, v)\}$, where E_i is the set of edges adjacent to some node i. If there is a pair of edges $(u, k), (v, k)$, then they are merged into a single edge (w, k) and their weights are summed. This process is repeated iteratively, until the graph has been coarsened to under 100 nodes.

An example is shown in Figure 3.3. The maximal matching from Figure 3.2b is used and the matched vertices are colored in. In this case every node in the graph is part of a matching. The merged vertices have the same color as its components.

This example also demonstrates why the graph coarsening approach is not a destructive one with regards to actually partitioning the graph. As we shall outline later, the goal of the partitioning stage is to find a partitioning with minimal edge cut. The vertices that are part of the edge cut are then included in the separator set, which completes the dissection. It is important to consider that the number of matched edges is highest in the most connected area of a graph. This is also the area of the graph where the edge cut is least likely to be created, because a high degree of connectivity likely increases the edge cut. Hence not much information is lost when these connected components are merged together into a large node, since they are not likely to participate in the edge cut anyways.

Since the goal of a coarsening round is to reduce the number of vertices in the graph, it is beneficial to have the most possible edges in each matching. A maximal matching is then the ideal solution, considering it maximizes the number of matches edges. However, the computational complexity of this operation is generally higher than that of finding the maximal matching. For this reason, MeTiS uses algorithms to find a maximal matching.

3.2.3. Graph partitioning

Having obtained a coarser graph, it is now easier to perform partitioning. In this step a minimum edge cut partitioning algorithm is used to generate a partitioning. Several partitioning methods are evaluated by METIS, with all performing somewhat similarly. The methods used for partitioning are as follows [26].

- **Spectral dissection** - a method where spectral information is used to partition the graph. The second largest eigenvector $\mathbf{y}$ is computed, whose values are used in a comparison. Every $y_j < r$ is assigned to one set, and the rest of the nodes to the other set. r is chosen as the weighted median of the values of y. In this way, the nodes are roughly split in half.
- **KL dissection** - a greedy method which starts with an initial partition and attempts to improve the partition (i.e. lower the edge cut) by iteratively swapping vertices from set to set. Once no improvement can be found the algorithm finishes.
- **Graph growing** - in this method we start from a random node and explore the graph in a breadth first manner until we have covered roughly half the nodes in the graph. Then the rest of the graph (which is unmarked) is put into the other set, to obtain a final partition. To make sure a larger portion of the search space is explored, it is possible to start multiple frontiers simultaneously. MeTiS usually chooses 10 frontiers. The partition can also be further refined by using it as input for KL dissection.
- **Greedy graph growing** - this method is similar to the last one, except at each BFS iteration the nodes on the frontier are ordered by increasing edge cut. Thus at each iteration a node with the lowest additional edge cut is added.

3.2.4. Uncoarsening

During the uncoarsening phase the smaller graph is expanded back to its original size. The partition is projected onto an uncoarsened graph by assigning expanded vertices to the same set as its parent. Suppose we have a set of vertices that were collapsed to a vertex v during the coarsening phase. Then these vertices are assigned to the same partition that v belongs to. Since the coarsening phase is an iterative process, the uncoarsening process is also iterative. This allows for iterative improvement of the partition during uncoarsening. MeTiS accomplishes this by applying the KL dissection algorithm during each phase of uncoarsening. A projected partition is used as the input to KL. Since this is already a good partition, KL is likely to converge in a few iterations. In the experience of the MeTiS researches this usually happened within three to five iterations.

4 Quantum Annealing

There are many more examples of NP- Hard or Complete problems that face researchers. While we can build heuristic algorithms using assumptions about specific underlying problems, there is also a need for algorithms that can efficiently solve difficult problems without having to rely on assumptions. These methods are called ”metaheuristics”, and we shall discuss several of them in this chapter.

The input of these algorithms is some optimization goal, usually expressed as some cost function f. A metaheuristic aims to find the global minimum of f, without prior knowledge of the solution landscape.

4.1. Simulated annealing

A well known concept in the field of metallurgy is that of annealing. This is a process whereby a solid that has been worked on is heated up, and subsequently allowed to slowly cool down. The solid, facilitated by a sufficiently high temperature, is able to rearrange its internal structure. After the annealing process, the solid has more favorable physical properties, having assumed a lower energy structural configuration.

As the name suggests, simulated annealing is emulating this physical process as a computational process. Metropolis et al. proposed a Monte Carlo method for simulating the evolution of a physical system, depending on a temperature T [39]. This simulation is applicable to any cost landscape, not just that of a physical solid. Hence it is possible to use this technique to solve any combinatorial optimization problem.

Optimization using simulated annealing The algorithm proposed by Metropolis et al. works as follows.

- Given the state of the system, a randomly generated disturbance is applied.
- If the energy difference ΔE between the current and disturbed system is negative , then the disturbance is accepted as the new state, and the process continues using this new state.
- If ΔE is positive, then the disturbed state is accepted depending on the probability distribution $\exp -\frac{\Delta E}{k_B T}$, where k_b is the Boltzmann constant, and T is the temperature of the system.
- Repeating these steps will eventually lead to a system close to thermal equilibrium. Once the system is sufficiently close to thermal equilibrium for a given T, the temperature is lowered and the disturbance process is repeated.
- The process is finished when the temperature has reached a stop criterion (for example, if the temperature has reached 0).

Figure 4.1 shows an example of an energy landscape suitable for solving with simulated annealing. The landscape has several energy troughs, separated by large peaks. Simulated annealing is able to escape a local minimum while the temperature is high. Over time the temperature is lowered, and states with a higher energy are less likely to be accepted, decreasing the chance the energy peaks are passed.

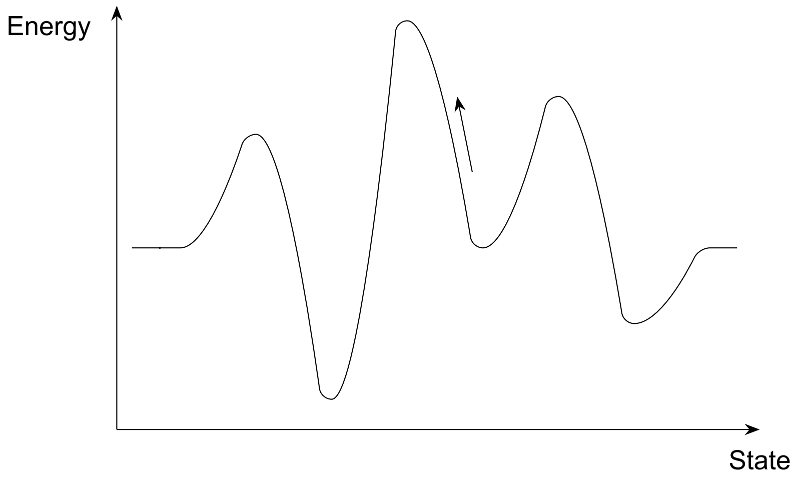


Figure 4.1: A high temperature allows the system to escape local minima. The arrow shows an example of the system moving to a higher energy, likely to be accepted under high temperature conditions.

Adiabatic condition Intuitively, the speed at which T is decreased has influence on the solution quality. The adiabatic condition is when T is varied such that the global optimum is found. Suppose we wish to find the minimum energy of some system represented by the Hamiltonian $\mathbf{H}$ (i.e. an Ising model). Geman et al. [22] showed that the annealing schedule that satisfies the adiabatic condition is of the form

$$T(t) = \frac{pN}{\log(\alpha t + 1)} \tag{4.1}$$

, where p is the entry-wise matrix norm of $\mathbf{H}$, N the number of variables in the system and α a constant relating to the energy gap between the ground and first excited state of the system. This is an important result because it shows the computational equivalence of SA and the optimization problems it is applied to. For an NP-Hard optimization problem, the logarithmic component in (4.1) means the equivalent SA process has exponential convergence time [40]. This proves that SA does not provide a polynomial time algorithm to solve non-polynomial problems.

4.2. Quantum annealing

Quantum annealing (QA) is a metaheuristic similar to simulated annealing in the sense that it explores the solution landscape driven by some fluctuation. In contrast to simulated annealing, QA uses quantum fluctuations instead of simulated thermal ones [21]. Considering in Figure 4.1 SA ”climbs” over energy peaks, an analogy for QA is that instead the system ”tunnels” through energy peaks.

We shall see that similarly to simulated annealing, quantum annealing also requires an annealing schedule. The quantum annealing schedule which satisfies the adiabatic condition is

$$\Gamma(t) = a(\delta t + c)^{-\frac{1}{2N-1}} \tag{4.2}$$

, where a and c are constants relating to the problem's spectral gap and N is the number of variables in the problem [40]. As with the schedule for SA, this schedule runs in exponential time, meaning it does not reduce NP problems to polynomial time. However, this schedule is faster than the one for SA. It is for this reason there is a lot of new research investigating the use of quantum annealing. The idea is that even though adiabatic evolutions take infeasibly long, the faster convergence time of QA compared to SA suggests QA is able to produce higher quality results than SA in the same amount of time.

The following section covers the working principles of quantum annealing, and how to express optimization problems such that they are applicable to be solved using a quantum annealer.

Lenz-Ising models A widely studied model in the field of statistical physics is that of the classical Ising model. While the Ising model has applications in many fields of study (such as chemistry, molecular biology and various disciplines in physics) we are primarily concerned with the magnetic application of the model. Consider a two dimensional lattice of N magnetic particles, arranged in a grid. The Ising model is used to represent the magnetic interactions between particles in the lattice. The Hamiltonian of the model is of the following form

$$H = H(\sigma) = -\sum_{<i,j>} J_{i,j}\sigma_i\sigma_j - \sum_i h_i\sigma_i \tag{4.3}$$

, where **J** defines the interaction strength between pairs of particles, and $\mathbf{h}$ defines the strength of an external field acting on each particle [10]. Here $\sigma_i \in \{-1, 1\}$ is the magnetic spin of particle i.

Finding a $\sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{N-1}\}$ such that $H(\sigma)$ is minimized is an NP-Hard optimization problem. As we shall see later in this section, it possible to express an arbitrary optimization problem in terms of an Ising model.

Transverse field Ising model Quantum annealers (like the D-Wave Advantage) make use of the transverse field Ising model (TFIM), which is defined as follows.

$$\hat{H} = \Gamma(t)\hat{H}_{TF} + \hat{H}_{\text{Ising}} = \Gamma(t)\sum_{i=1}^{N}\sigma_i^x + \sum_{i=1}^{N} h_i\sigma_i^z + \sum_{<i,j>} J_{i,j}\sigma_i^z\sigma_j^z \tag{4.4}$$

, where σ^x and σ^z are the Pauli spin operators, which are quantum operators corresponding to a particle's spin in the x and z axis respectively [40]. Their eigenvectors correspond to the unit spin on their axis. For example, the eigenvectors of σ^z are $\begin{bmatrix}0\\1\end{bmatrix}$ and $\begin{bmatrix}1\\0\end{bmatrix}$.

What sets the TFIM apart from the classical Ising model is the introduction of the transverse term $\hat{H}_{TF}$, which is defined along the x-axis. The spin operators along different axes are non-commuting. This means it is physically impossible to apply these operators simultaneously [44]. Observing the spin of a particle along one axis puts the particle in a superposition along the other axes. This is a fundamental property of quantum systems. It is now no longer sufficient to use classical methods to study the properties of this system.

The annealing process The goal of the annealing process is to bring our quantum system into a state which corresponds to the ground state of $\hat{H}_{\text{Ising}}$. Without intervention, this does not happen automatically. We must introduce a term which induces an exploration of the solution landscape. In the TFIM, this is achieved using the transverse component $\hat{H}_{TF}$. At the start $\Gamma(t = t_0) = K, K >> 1$. Now σ^x is the dominating component in the system, putting the z-axis spins in a superposition. This state is considered the magnetic disordered state, because of the uncertainty of the z-aligned spins. It has a ground state which is aligned along the x spin axis [38].

Over the course of the annealing process, $\Gamma(t)$ is reduced, increasing the relative strength of the $\hat{H}_{\text{Ising}}$ component [1]. The system reaches a point at which $\hat{H}_{\text{Ising}}$ is the dominant component. In this state the contribution of the transverse field is minimal, and the ground state of the system must be aligned along the z spin axis. This is called the ordered state. In both the ordered and the disordered state the presence of a non-commuting component causes disturbances. These disturbances allow the system to explore the energy landscape.

4.2.1. Modeling problems as Ising models

Using the quantum annealing process we can find the ground state of an arbitrary Ising Hamiltonian $\hat{H}_{\text{Ising}}$. The corresponding state vector is the lowest energy configuration of the classical Ising model H_{Ising}. As we have seen before, finding this configuration is NP-Hard. We can transform other combinatorial optimization problems to its equivalent Ising model, and then use quantum annealing to find its solution. This is accomplished by changing the $J_{i,j}$ values in the Ising Hamiltonian such that a ground state $|s\rangle$ satisfies $\arg\min_s f(s)$. The translation of an optimization problem to its equivalent Ising model

[1]In the D-Wave quantum annealer not only is the strength of $\hat{H}_{TF}$ decreased, but $\hat{H}_{\text{Ising}}$ is simultaneously increased. For our theoretical analysis this does not matter, but in the presence of thermal fluctuation, increasing the strength of the Ising component increases the reliability of the computation.

requires some work, and the formulation for nested dissection in this form is the primary contribution of this work.

Quadratic unconstrained binary optimization In the Ising Hamiltonian the state vector $|s\rangle$ has $s_i \in \{-1, 1\}$. While this is convention for its physical interpretation, it is often more intuitive to be able to express states in terms of binary variables. To this end we construct a new state vector $|b\rangle$ with domain $b_i \in \{0, 1\}$. This variant of the Ising model is called quadratic unconstrained binary optimization (QUBO) model. For all intents and purposes they are equivalent, and converting between the two representations is trivial using the following equations:

$$\begin{aligned} b_i &= \frac{s_i + 1}{2}, \\ s_i &= 2b_i - 1 \end{aligned} \tag{4.5}$$

Example Suppose we are a company that needs to decide which projects to fund. Each project has an associated cost and profit, and the company is on a limited budget. The company seeks to maximize the profit, while staying within the bounds of their allotted budget. Now we very clearly have an optimization goal, and constraints. This is exactly how we shall later develop the QUBO formulation of the minimum vertex separator problem of nested dissection. For now, we can transform our simple budget allocation problem into a QUBO.

To do so we define three binary values x_1, x_2, x_3, which correspond to the three projects our fictional company can invest in. Next we can define our optimization goal by writing the expression $p_1x_1 + p_2x_2 + p_3x_3$, where each p_i is the profit corresponding to each p_i. For the constraint, we must define an expression which evaluates to a large number when the summed cost is below or above b, our budget. To achieve we can leverage a quadratic formulation, as follows: $(c_1x_1 + c_2x_2 + c_3x_3 - b)^2$. Now if $(c_1x_1 + c_2x_2 + c_3x_3)$ is significantly above or below b, the quadratic expression will evaluate to a large number.

Adding up the optimization goal and constraint gets us the complete formulation of our problem as a QUBO:

$$H(x_1, x_2, x_3) = -(p_1x_1 + p_2x_2 + p_3x_3) + (c_1x_1 + c_2x_2 + c_3x_3 - b)^2$$

We can transform this expression into a matrix with $\mathbf{Q}_{i,j}$ like we previously discussed by expanding the quadratic expression and collecting similar terms. Note that because they are binary variables, every $x_i^2 = x_i$.

$$\begin{aligned} H(x_1, x_2, x_3) = -(p_1x_1 + p_2x_2 + p_3x_3) + c_1^2x_1 + c_2^2x_2 + c_3^2x_3 + b^2 \\ - bc_1x_1 - bc_2x_2 - bc_3x_3 + 2c_1c_2x_1x_2 + 2c_1c_3x_1x_3 + 2c_2c_3x_2x_3 \end{aligned}$$

$$H(x_1, x_2, x_3) = x_1(c_1^2 - bc_1 - p_1) + x_2(c_2^2 - bc_2 - p_2) + x_3(c_3^2 - bc_3 - p_3) + 2c_1c_2x_1x_2 + 2c_1c_3x_1x_3 + 2c_2c_3x_2x_3 + b^2$$

In this expanded form we can collect the quadratic and linear terms into the matrix **Q** and express the entire formulation as $H(\mathbf{s}) = \mathbf{Q}\mathbf{s} + c$, where $\mathbf{s} = [x_1 \quad x_2 \quad x_3]$, and

$$\mathbf{Q} = \begin{bmatrix} c_1^2 - bc_1 - p_1 & c_1c_2 & c_1c_3 \\ c_2c_1 & c_2^2 - bc_2 - p_2 & c_2c_3 \\ c_3c_1 & c_2c_3 & c_3^2 - bc_3 - p_3 \end{bmatrix}$$

, and $c = b^2$. Finding an **s** that minimizes the function $H(\mathbf{s}) = \mathbf{Q}\mathbf{s} + c$ solves our optimization problem. Making these matrices by hand is often quite a cumbersome task. Fortunately, D-Wave, the manufacturer of the largest quantum annealer currently available, provides some Python utilities that make the construction of QUBOs easier. This will be elaborated upon in Chapter 5. It is important to understand that despite whatever representation one might consider using, the quantum annealer still only accepts problems described as an interaction matrix, like the one we just developed.

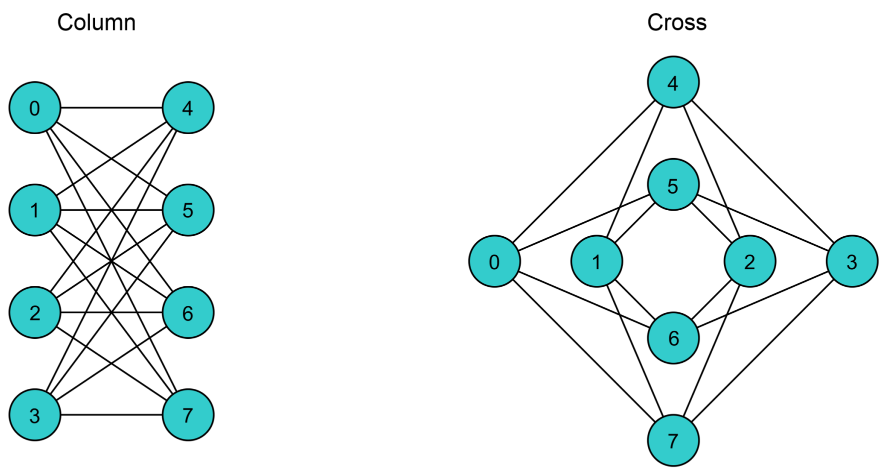


Figure 4.2: An example of a QPU topology.

4.2.2. Discrete integer encoding

As alluded to in previous sections, while qubits are able to take a superposition of multiple values at once, their values still collapse to either a 1 or 0 upon measurement. This means (as is also the case with classical computers) that if we wish to encode values beyond 1 and 0 in a logical group (such as a discrete integer), we must devise some mathematical scheme to handle this.

One might quickly arrive at encoding integers the same way they are on classical computers, by representing them in base-2 binary values. The number of bits required for storing m different values is $\log_2 m$, which is very good scaling. However, logical variables are used in a different way in classical and quantum computers. In classical computers when we think about the efficiency of a representation, we consider what impact this has in terms of Random Access Memory, both in the amount of storage it takes and the access patterns.

On a quantum annealer, the situation differs. Indeed, the number of qubits is limited, but more pressingly the number of couplers (physical interactors between qubits) is also limited. This presents a different limitation on the encoding we use compared to classical computers. One of the limiting factors when it comes to coupling is that physically the interactions are only able to be of the second order. This means the variables can have at most a quadratic interaction. Binary encoded variables have higher order interactions, which can be converted to quadratic interactions at the cost of extra auxiliary qubits. It is more intuitive to use different encoding schemes which are more tailored to the quantum annealing application.

In Chapter 5 we present two common QUBO encoding techniques, namely One Hot [42] and Domain Wall [8] encoding.

4.2.3. Minor embedding

As with most matrices, the interaction matrix of QUBO and Ising problems can also be interpreted as a graph. When we do this for an arbitrary problem it quickly becomes apparent that these graphs can become quite connected. For a simulated solver this is not necessarily an issue, but this does become problematic when we wish to evaluate these problems on a physical system, such as on a quantum annealer. This is because a physical system is also constrained by a physical topology, and at the time of writing these topologies do not allow for unbounded connectivity between all nodes.

Instead, topologies of input problems are required to be mapped on the underlying topology of the quantum annealer [12]. Figure 4.2 shows an example of one of these topologies. In this case it’s the Chimera unit cell topology, which was used by the D-Wave 2000Q systems. The edges in this graph represents a physical coupler able to couple qubits together. That gives rise to interaction as modeled by the Ising model. Chimera qubits have a nominal length of four, meaning they are connected to four different qubits in the same unit cell. They have a degree of six, which also includes their connection to qubits outside of their unit cell. Our problem Ising models may not necessarily be limited to a maximum degree of six, and often they are not. Furthermore, the problem Ising models may be of a shape which is incompatible with the annealer topology. This is where the process of minor embedding emerges.

Minor embedding is the process of mapping these high degree Ising models onto the lower degree QPU topology. Need for high degree nodes is simulated by ”chaining” qubits together. This means they are coupled in such a way that they are strongly incentivized to hold the same value. The strength of this chain is called the chain strength, and is a parameter that can be tuned when submitting a problem to D-Wave. Minor embedding in general is NP-Complete, and therefore D-Wave uses a heuristic minor embedder for most problems.

The embedding of the problem graph onto the QPU topology can have a major impact on the solution quality. Some patterns can be avoided in the problem formulation to make minor embedding easier. In general, the more variables a particular variable interacts with, the more difficult the embedding tends to be. For example, all to all connections are particularly costly.

5 Methods

To solve the minimum vertex separator problem on a quantum annealer, we have to express the problem as an Ising model. Just as in the example QUBO presented in Chapter 4, we start by identifying the goals and constraints of the problem, and subsequently express these in some quadratic form. Unlike the example, we are dealing with a problem that involves discrete options (i.e. which set does the node belong to?), and thus we have to use a binary encoding method. For the purpose of investigation, we develop the QUBO in two different encoding methods, the One Hot and Domain Wall methods.

To avoid repetition, we first express the goals and constraints in an encoding agnostic manner, through the Discrete Quadratic Model (DQM). In essence, this formulation works on discrete variables instead of binary ones, removing the need for an encoding technique. Later, after having presented the workings of both encoding techniques, we translate the DQM representation to a binary one.

Finally, we examine what the inclusion of a quantum step means for a generalized finite element solver.

5.1. Goals and constraints

As discussed in Chapter 4, the natural language of quantum annealers is that of an Ising model, often expressed as a quadratic unconstrained binary optimization (QUBO) problem. However, not all problems can be mapped directly to a binary model. An intermediate model operating on discrete variables is a useful abstraction to express goals and constraints, irrespective of their final binary encoding. To this end we use the discrete quadratic model (DQM) to express the goals and constraints corresponding to nested dissection. After the DQM description, the model is encoded as a QUBO problem using the ”One hot” and ”Domain wall” techniques.

5.1.1. Discrete Quadratic Model

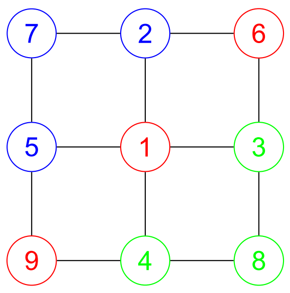
A Discrete Quadratic Model (DQM) is a method of expressing an optimization problem. The most general DQM is as follows

$$H_{DQM} = \sum_{i} A_{(i)}(d_i) + \sum_{i \geq j} B_{(i,j)}(d_i, d_j). \tag{5.1}$$

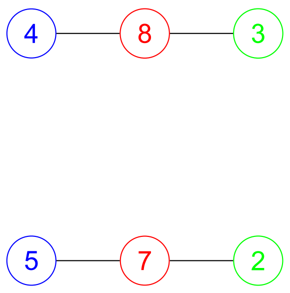
Here each d_i is a discrete variable, and each $A_{(i)}$, $B_{(i,j)}$ are real valued functions operating on these variables. In the case of nested dissection, after a dissection each node is assigned to one of three sets A, B, S. We can encode this in the DQM by introducing a variable $d_i \in \{A, B, S\}$ for each node in our graph.

It is also possible to express a DQM as a set of binary variables, which will make the encoding of our goals and constraints more intuitive. We can represent a single d_i as a set of binary variables $x_{i,\alpha}$, where α refers to the variable's state. More specifically, each $x_{i,\alpha}$ is such that

$$x_{i,\alpha} = \begin{cases} 1, & d_i = \alpha \\ 0, & \text{otherwise} \end{cases} \tag{5.2}$$



(a) First dissection.



(b) Second dissection steps.

Figure 5.1: Example of a dissection that produces non-optimal ordering on a 3x3 grid graph.

.

Using this notation it is easy to express complex Hamiltonians based on the assigned set to each node. For example, to penalize nodes being assigned to set B, we can construct the Hamiltonian

$$H_{EX} = \sum_{i=0}^{n-1} x_{i,B}.$$

.

5.1.2. Minimal separator goal

As described in Chapter 3, the performance of nested dissection is directly correlated with the size of the separator set S in each step. To this end we assign a linear penalty for each node i assigned to the separator set S,

$$H_{MS} = \sum_{i=0}^{n-1} x_{i,S}. \tag{5.3}$$

5.1.3. Minimal edge cut goal

George [23] shows that on various graph types (including grid graphs) it is possible to construct a nested dissection ordering which is optimal in terms of fill-in. However, not every nested dissection ordering is an optimal one. Figure 5.1 shows valid dissection decisions on a 3x3 grid graph that produces a non-optimal ordering. The optimal solution has the first dissection along the middle of the graph, instead of the diagonal.

Without additional work, the optimal and non-optimal solutions are energetically equivalent to a quantum annealer. To solve this, we identify that the difference between the two solutions is the number of edges intersected by the separator set. The dissection on the diagonal cuts 8 edges, while the vertical/horizontal dissection only cuts 6 edges.

To minimize the edge cut between S and sets A and B, we penalize every edge connecting S with A or B. This is achieved by leveraging the properties of binary variables. For a given edge $e_{i,j} \in E$ the expression $x_{i,A}x_{j,B} = 1$ iff the edge connects a node in A to a node in B. Since a node can be in either A or B, for a given edge $e_{i,j}$, it is required to consider both cases $n_i \in A$ and $n_i \in B$. This can be expressed in DQM form with the following Hamiltonian.

$$\sum_{\{i,j\}\in E} (x_{i,A}x_{j,S} + x_{i,B}x_{j,S} + x_{j,A}x_{i,S} + x_{j,B}x_{i,S}). \tag{5.4}$$

5.1.4. Set independence constraint

As described in Chapter 3, we only have a valid separator when we there are no edges connecting set A and B. The approach outlined in the previous subsection is used again. Instead of penalizing edges between S and A or B, we penalize edges between A and B. This gives rise to the following Hamiltonian

Set	One hot
A	[1 0 0]
B	[0 1 0]
S	[0 0 1]

Table 5.1: Encoding scheme with one hot encoding

$$H_{IS} = \sum_{\{i,j\}\in E} (x_{i,A}x_{j,B} + x_{i,B}x_{j,A}). \tag{5.5}$$

5.1.5. Balance soft constraint

Only applying the set independence constraint to the minimization of the separator set is insufficient. The ground state of this resulting Hamiltonian will be one where every node is assigned to either set A or B. This solution minimizes the size of the separator, and ensures no edges connect set A and B.

An improved encoding would also penalize imbalance between A and B. This improves the solution quality two-fold. Firstly, it prevents the model placing all nodes in the same set. Secondly, balanced set sizes help distribute computational load in multithreaded situations. Indeed it is possible to consider balancing sets A and B as another optimization goal. However, since the lack of a balancing Hamiltonian results in the model producing invalid solutions, we consider it a constraint instead.

The constraint is satisfied when $|A| = |B|$, any deviation from this equality should be penalized. Therefore, in the following Hamiltonian, we square the difference between $|A|$ and $|B|$, becoming 0 when $|A| = |B|$, otherwise it introduces a penalty.

$$H_{BC} = \left\{\sum_{i=0}^{n-1} x_{i,A} - x_{i,B}\right\}^2. \tag{5.6}$$

5.2. One hot encoding QUBO

One hot encoding is a straightforward technique for encoding discrete integer problems in binary form. For a discrete variable x with m different values, we create a group b of m bits. The first bit b_0 is 1 if and only if the variable has a discrete value of 0. This pattern is repeated for every possible value of x. In the case of nested dissection we have $m = 3$. The encoding for each set is presented in Table 5.1.

5.2.1. Encoding constraint

For there to be a valid one hot encoding, only one bit in the bitset is allowed to be 1. This constraint must be encoded in the QUBO. Intuitively, there is a valid one hot encoding when the sum of all bits is equal to 1. This is described by the following Hamiltonian.

$$H_{OH} = k\left(\sum_{i=0}^{m-1} b_i - 1\right)^2. \tag{5.7}$$

5.2.2. Goals and constraints

Translating the DQM representation to a one hot encoding QUBO is trivial. We can replace every x_α, where $\alpha \in \{A, B, S\}$ with a corresponding b_i, where i is the corresponding bit index according to Table 5.1. All translated goals and constraints are summarized in Table 5.2.

5.3. Domain wall encoding QUBO

Domain wall encoding is a more recent technique to encode discrete integer problems as a QUBO. The technique is inspired by the equivalent magnetic effect, where a domain wall is defined to be the interface between two magnetic moments. For our purposes, a domain wall can be defined to be present when two adjacent binary variables i and $i + 1$ have unequal values. Suppose we wish to

Goal/constraint	DQM	QUBO
Minimal separator goal	$\sum_{i=0}^{n-1} x_{i,S}$	$\sum_{i=0}^{n-1} b_{i,2}$
Minimal edge cut goal	$\sum_{\{i,j\}\in E}(x_{i,A}x_{j,S} + x_{i,B}x_{j,S} + x_{j,A}x_{i,S} + x_{j,B}x_{i,S})$	$\sum_{\{i,j\}}(b_{i,0}b_{j,2} + b_{i,1}b_{j,2} + b_{j,0}b_{i,2} + b_{j,1}b_{i,2})$
Balance constraint	$\{\sum_{i=0}^{n-1} x_{i,A} - x_{i,B}\}^2$	$\{\sum_{i=0}^{n-1} b_{i,0} - b_{i,1}\}^2$
Set independence constraint	$\sum_{\{i,j\}\in E}(x_{i,A}x_{j,B} + x_{j,A}x_{i,B})$	$\sum_{\{i,j\}\in E}(b_{i,0}b_{j,1}+b_{j,0}b_{i,1})$

Table 5.2: Goals and constraints expressed in DQM and QUBO form for one hot encoding.

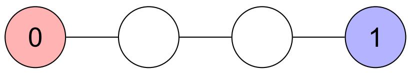


Figure 5.2: One dimensional Ising chain encoding a logical variable of $m = 3$. The red and blue node represent the virtual binary variables b_{-1} and b_{m-1} respectively.

represent a variable x of size m. We construct a chain of $m - 1$ binary variables, the variable's logical value is encoded in the location of the domain wall in this chain. The chain is visualized in Figure 5.2. For a chain of binary variables $b = b_0, b_1, b_2, \cdots, b_{m-2}$, we introduce two "virtual" binary variables $b_{-1} = 0$ and $b_{m-1} = 1$. They are not present in the QUBO, however, defining the boundary conditions as such simplifies the mathematics.

To encode the three different sets required for nested dissection, we can extend Table 5.1 to Table 5.3. As can be seen each set is encoded as the position of a domain wall in a chain of binary variables. For set A the wall is present between b_1 and $b_2 = 1$, for set B between b_0 and b_1, etc.

5.3.1. Encoding constraint

As with one hot encoding, domain wall encoding requires a constraint to be added to the QUBO to keep variables in a valid state. For domain wall encoding there is only a valid state when there is a single domain wall present in the variable chain. When using spin variables $s_i \in \{-1, -1\}$ this constraint can be neatly described using the Hamiltonian

$$H_{DW} = -\kappa \sum_{i=-1}^{m-2} s_i s_{i+1}. \tag{5.8}$$

Intuitively this formulation can be substantiated by considering that each present domain wall increases the Hamiltonian's energy. The boundary conditions $s_{-1} = -1$, $s_{m-1} = 1$ enforces the presence of at least a single domain wall. The Hamiltonian enforces that no other domain walls can be present.

To translate between spin variables and binary variables (required for the QUBO formulation), we can use the substitution $s_i = 2b_i - 1$. The Hamiltonian using binary variables is

$$H_{BDW} = -\kappa \sum_{i=-1}^{m-2} 1 + 4b_i b_{i+1} - 2b_i - 2b_{i+1}. \tag{5.9}$$

For the case when $m = 3$ (we have three sets) this equation simplifies to

Set	One hot	Domain wall
A	[1 0 0]	0 [0 0] 1
B	[0 1 0]	0 [0 1] 1
S	[0 0 1]	0 [1 1] 1

Table 5.3: Encoding scheme extended with domain wall encoding

$$H_{BDW} = -\kappa\left(1 - 4b_0 + 4b_0 b_1\right). \tag{5.10}$$

Each node in our graph is assigned to a set, hence the final Hamiltonian is summed over all nodes $i \in \{0, 1, 2, \cdots, n\}$, with each being represented by a set of binary variables $b_{i,0}, b_{i,1}$

$$H_{BDW} = -\kappa \sum_{i=0}^{n-1} 1 - 4b_{i,0} + 4b_{i,0}4b_{i,1}. \tag{5.11}$$

Constant term in encoding constraint One might notice that in (5.11) there is a constant term, that when taken out of the sum is equal to $-n\kappa$. This constant term is not obviously present in the spin variable formulation of the encoding constraint ((5.8). However, if we calculate the value of H_{DW} when, $m = 2$, $s_0 = s_1 = -1$, then we find it's equal to $-\kappa$. Indeed, if we work out the value for any set of valid $m = 3$ domain wall encodings we find $H_{DW} = -\kappa$. This phenomenon is inconsequential from the perspective of an annealer, since a violation of the domain wall is still energetically higher.

Practically, however, the presence of a constant term does result in confusion when interpreting the results. When implementing the spin variable formulation, every solution will have a large offset if not compensated. With the binary formulation, the constant is explicitly stated in the formulation. In our implementation of the binary model the constant is deliberately not included to make the energy values easier to compare with one-hot encoding.

5.3.2. Goals and constraints

Goals and constraints expressed in terms of DQM variables can be translated to domain wall encoding using

$$x_\alpha = b_\alpha - b_{\alpha-1}, \tag{5.12}$$

subject to the boundary conditions $b_{-1} = 0$, $b_{m-1} = 1$. Apart from the encoding constraint, the goals and constraints are the same ones as described in Section 5.1.

Determining the form of the constraints in QUBO form is more work for domain wall than it is for one-hot encoding. To encode in one-hot form it is as simple as replacing the discrete variable with the corresponding binary one. However for domain wall encoding one must perform considerable amounts of algebra to reach constraint expressions.

Goal/constraint	DQM	QUBO
Minimal separator goal	$\sum_{i=0}^{n-1} x_{i,S}$	$\sum_{i=0}^{n-1} b_{i,0}$
Minimal edge cut goal	$\sum_{\{i,j\}\in E}(x_{i,A}x_{j,S} + x_{i,B}x_{j,S} + x_{j,A}x_{i,S} + x_{j,B}x_{i,S})$	$\sum_{\{i,j\}}(b_{i,0} + b_{j,0} - 2b_{i,0}b_{j,0})$
Balance constraint	$\left\{\sum_{i=0}^{n-1} x_{i,A} - x_{i,B}\right\}^2$	$\{n + \sum_{i=0}^{n-1} b_{i,0} - 2b_{i,1}\}^2$
Set independence constraint	$\sum_{\{i,j\}\in E}(x_{i,A}x_{j,B} + x_{j,A}x_{i,B})$	$\sum_{\{i,j\}\in E}(b_{j,1} - b_{j,0} + b_{i,1} - b_{i,0} - 2b_{i,1}b_{j,1} + b_{i,0}b_{j,1} + b_{i,1}b_{j,0})$

Table 5.4: Goals and constraints expressed in DQM and QUBO form for domain wall encoding.

5.4. Hyperparameters

One thing that has not been mentioned so far is the concept and importance of hyperparameters. So far we have only defined the goals and constraints of our QUBO (which is indeed the most crucial step), but we have not yet summed them together to form the final formulation. When summing the individual components, we have the choice to apply a multiplier to each goal and constraint. These multipliers are more commonly referred to as hyperparameters. It turns out that the choice of hyperparameters can have a significant impact on the quality and stability of a solution. This is also the case for the QUBOs defined for nested dissection.

Currently there is no known way to efficiently optimize the hyperparameters of a QUBO, and choices often rely on intuition. As we see in Chapter 6, the optimal choice of hyperparameters often depends on the problem structure. Additionally, there can be unexpected interactions between different goals and constraints that make a general optimization strategy difficult to achieve. The only general optimization strategy is that of a plain parameter sweep. The nested dissection QUBO has five separate hyperparameters, which makes for a considerable solution space, especially considering multiple problem sets. This makes finding the optimal hyperparameters, or even patterns that might help us choose them, difficult.

Nonetheless, as a baseline, we choose the hyperparameters such that the constraints weigh heavier than the optimization goals. The reason for this is that in our problem statement the constraints are hard requirements. Without following the constraints, the resulting solution is useless, and subsequent iterations of nested dissection are unable to complete. Thus, we reflect this requirement in the hyperparameters, because a valid solution with high separator size is still better than an invalid solution with low separator set size.

5.5. Formulation complexity

The formulation of the QUBO is in essence a data transformation. As with any transformation, there is an algorithmic complexity associated with performing this task. For both the one hot and domain wall encoded QUBOs we have several Hamiltonians that sum over either n or $|E|$, where E is the set of edges in the graph. Hence in total in at least the first dissection step the QUBO formulation complexity is $O(n + |E|)$. We can see that this is the same for every other dissection step as well. The $k + 1$th dissection step involves the creation of twice the amount of QUBOs solved in the kth step. Each of these formulations use half the amount of nodes of the previous step. So while the number of variables in each QUBO changes, the overall amount of used variables stays the same. Assuming each step approximately dissects the graph evenly, the full nested dissection completes in $\log_2 n$ steps. Hence the total complexity of the QUBO based nested dissection algorithm is $O((n + |E|) \log_2 n)$.

5.6. Algorithm workflow

The addition of a quantum step impacts the workflow of a finite element solver. Here we examine the traditional and quantum solver pipeline. The traditional pipeline is shown in Figure 5.3. It is important to highlight that while matrix preprocessing is an important part of the pipeline, it does not need to run for every iteration. An ordering is effective regardless of the values that occupy the matrix. Hence if the input problem only changes numerically, but not structurally, then we can reuse the ordering, without having to recompute the preprocessing. This is quite common in finite element applications, where the structure of the matrix is determined only by the discretization of the function domain. The particular function does not impact the matrix structure.

The matrix preprocessing pipeline changes when a quantum step is introduced. There is more preprocessing involved than compared to the classical approach. In particular, the QUBO needs to be constructed. Then the problem is submitted to D-Wave, where it is first minor embedded and then annealed. Afterwards, we use a post-processing step to optimize the returned solution even more, before decoding it and using it in the ordering.

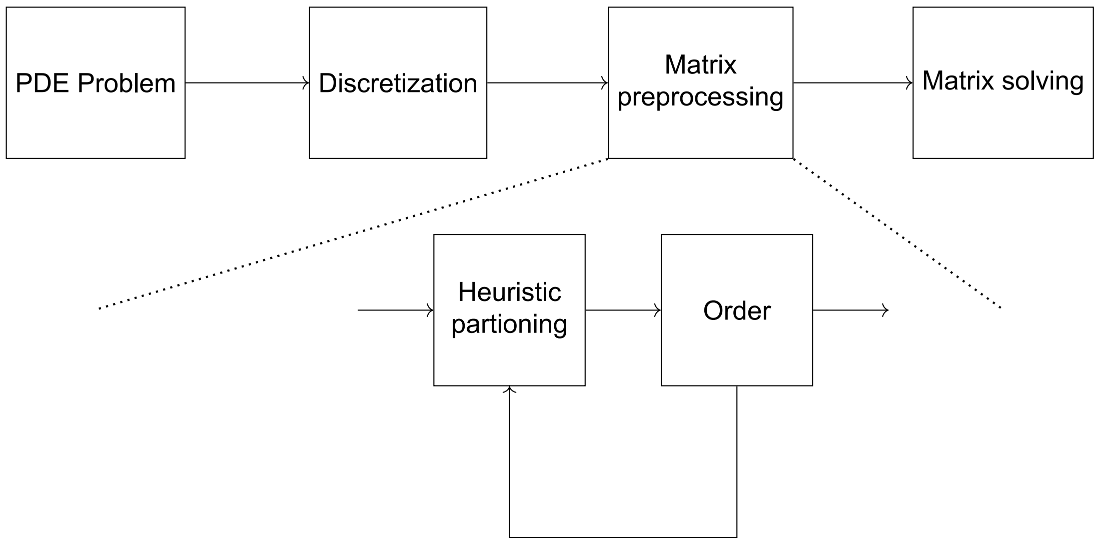


Figure 5.3: Traditional solver pipeline

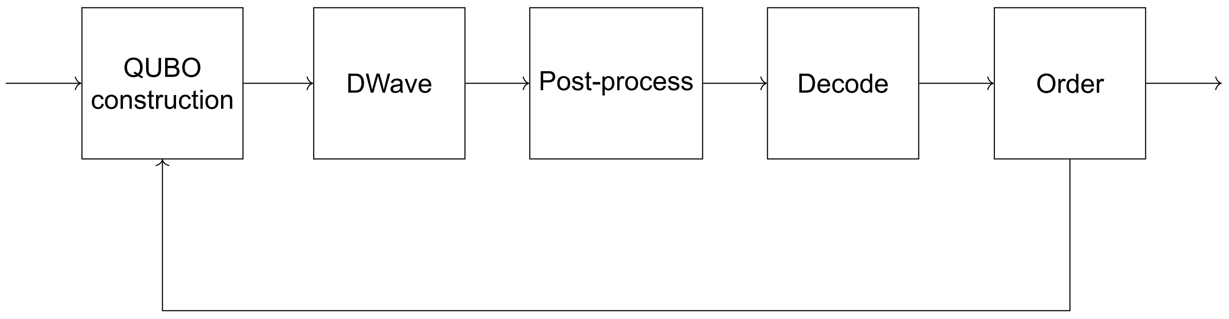


Figure 5.4: Matrix solver step with quantum annealing.

6
Results

6.1. Datasets

6.1.1. Synthetic datasets

Synthetic datasets are ones that are computer generated. Below is a summary of every synthetic dataset used, and its form.

Grid graph As the name implies, a grid graph is one that has the form of a grid. Several examples have already been presented. These graphs primarily emerge from finite element applications, and is therefore an interesting graph form to examine. Moreover, the nested dissection heuristic was developed principally for efficiently ordering grid graphs. Hence, it is interesting to consider the performance of the QUBO on this dataset.

Wheel graph A wheel graph is a graph where one node is surrounded by a collection of connected nodes. An example is presented in Figure 6.1. This graph is interesting because an optimal solution is trivially found by both minimum degree and nested dissection. Additionally, the dissection is quite simple, as any cut down the middle of the wheel, no matter the size, is the optimal dissection.

Erdős-Rényi graph An Erdős-Rényi graph is a type of randomly sampled graph, it is sampled in a binomial manner. For this reason it is also known as a binomial graph. It has two parameters, n and p, which is why often the graph will be referred to as a $G_{n,p}$ graph. To generate a $G_{n,p}$ graph, we fix the set of vertices $V = \{1, 2, ..., n\}$ and the number of edges $N = \binom{n}{2}$. The set of all edges is then $E = \{e_0, e_2, ..., e_N\}$. Then, we sample each edge with a probability of p [17].

Figure 6.2a shows an example of what an $n = 10$, $p = 0.3$ graph looks like. For the rest of the experiments $p = 0.3$, and n is varied.

Geometric graph A random geometric graph is a random graph where the points are generated inside some geometric space, and edges are generated to connect nodes which are in a specific radius of each other. The nodes are placed in a k dimensional space, with probability p. These nodes are connected with each other if they are in radius r of each other [43]. Figure 6.2b shows an example geometric graph. Notice how the geometric method naturally introduces distinct components in the

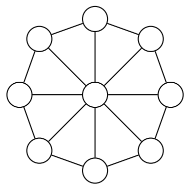
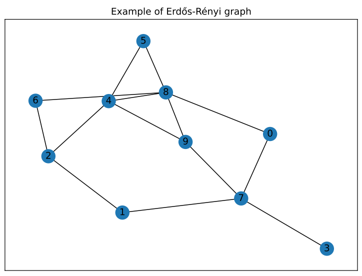
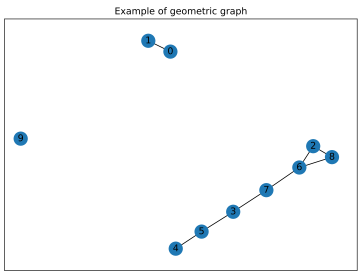


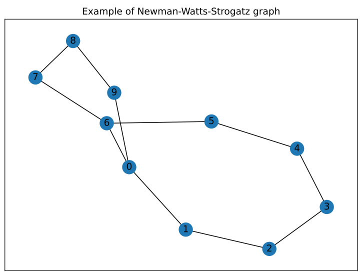
Figure 6.1: A wheel graph.



(a) Example of an Erdős–Rényi graph.



(b) Example of a geometric graph



(c) Example of a Newman Watts Strogatz graph

Figure 6.2: Examples of the randomly generated graphs used in experiments.

graph. In every experiment we use $r = 0.3$ and $k = 2$. The example is generated using $n = 10$. It is possible for this algorithm to produce disconnected graphs. Our formulation of the nested dissection algorithm is agnostic to disconnections within the graphs.

Newman Watts Strogatz graph The Newman Watts Strogatz method of generating a random graph is one that aims to simulate a “small-world” scenario, with a high degree of clustering. To do this, all n nodes are initially connected in a cycle. Then each node is connected to its k nearest neighbors. Finally shortcuts are introduced by introducing a new edge (u, w), where w is a random node, for each edge (u, v) with probability p. For this graph a k, p and n must be chosen. Every experiment varies n, and uses $p = 0.3$. The nearest neighbor parameter $k = \max(\frac{n}{10}, 2)$ [41].

6.2. On recursive QUBO application

Regardless of solving a QUBO using simulated or quantum annealing, every solver will produce a set of solutions, with varying solution energies. Assuming a correct QUBO formulation, the lowest energy solution should correspond to the optimal solution. While this assumption is easy to verify for certain simple problems, in more complicated solver pipelines this becomes less obvious. Such is also the case for nested dissection, where our QUBO serves to replace a heuristic, which itself is used to guide another heuristic.

Another important point of analysis is that of hyperparameters. The specific tuning of hyperparameters is not immediately obvious, neither is the effect they have on the overall algorithm performance (i.e. how much fill-in is produced?). Just like solution energy, it is not obvious what the higher order effects are of different hyperparameters. For example, loosening the balance constraint might degrade the immediate performance of the QUBO in the first iteration of ND, and subsequently lead to more favorable solutions in the higher order iterations, ultimately leading to a lower overall fill-in.

It is therefore useful to be able to examine how well solution energy correlates with algorithm performance. Due to the recursive nature of nested dissection, where a QUBO is applied multiple times, this is not a trivial task. In our pipeline every invocation of the QUBO depends on the solution that preceded it. This makes visualization and analysis of the results tedious, as it is impossible to predict the effects of the first solution on the subsequent solutions.

6.2.1. Tree representation of solutions

To make the visualization and analysis of solutions more accessible, we organize the set of all solutions in a tree structure. This tree structure is illustrated in Figure 6.3. Every node represents a sample taken from a solver, containing the different set assignments and the recursion level at which the sample was taken. A solution instance is a path from a single root node to a single leaf node. Paths are defined using hashes. Each node contains a ”parent” attribute, which is a hash value that corresponds to its parent’s hash value. The hash is calculated considering all node parameters except its hash value.

A tree is a natural way of expressing the steps of nested dissection, as each recursion step creates two new ”branches” (at every step the ND is invoked again twice). The solution space can then be explored by examining every possible path in the tree. Each new path is a new solution, for which we can calculate the fill-in. This technique is used in various experiments presented in this section.

The procedure for generating a complete nested dissection order from this tree is as follows.

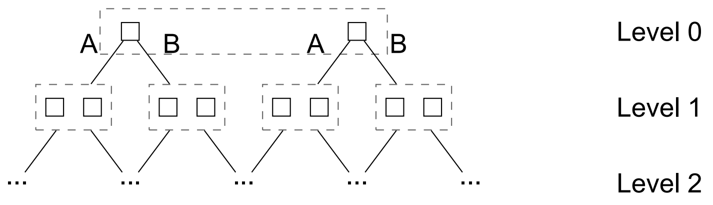


Figure 6.3: Tree representation of nested dissection results used for evaluating the impact of different dissection decisions.

1. Append each node at Level 0 to a stack, initialize a corresponding partial ordering for each node.
2. Pop a node from the stack, this is the current node.
3. Append the current node's separator set contents to the partial ordering.
4. If the partial ordering's length is equal to the total amount of vertices in the graph, this partial ordering becomes a full ordering and is appended to the set of orderings. This also implies that the current node is a leaf node.
5. For every child on the left and right side, push the child node to the stack. If there are multiple children on either side, then for every child that is not the first one, initialize a new partial ordering. This ensures that a new path also results in a new ordering.
6. Return to step 2.

6.3. Experimental setup

In this section we describe the various experiments that were performed. The main performance indicator of the QUBO formulation is how much fill-in the suggested ordering results in. We also investigate how well the QUBO scales, which shows how large problem size can get before physical quantum computers are unable to solve the problem. Scaling is also important because in general smaller QUBOs are easier to solve. Finally, we also present various experiments related to exploring the hyperparameter search space.

6.3.1. Performance

Fill-in To evaluate the performance of the QUBO, we use it to produce a nested dissection ordering for several different problem sets and sizes. We run this experiment on every synthetic dataset described in Section 6.1.1. The experiments are run on graph sizes 10, 20, 50, 100 and 150. Most of the experiments are completed using simulated annealing. The reasons for this are two-fold. Firstly, current quantum annealers are time shared, and running many large scale nested dissection experiments on a physical quantum annealer is not cheap. For this reason we limited the quantum annealing based experiments to only grid graphs. Secondly, the problem sizes that are able to run on quantum annealers are inherently limited in size. Simulated annealing does not suffer from these limitations, as classical computers are currently able to handle much larger problem sizes. Additionally, classical computation resources are widely available, in contrast to quantum computing resources.

For the quantum annealing sampler we use the TNO quantum toolbox to wrap the D-Wave provided QPU (quantum processing unit) sampler. Included in this wrapper is a post-processing tool that uses gradient descent to greedily optimize the results return by the quantum annealer. We enable this for this experiment.

Since we are testing performance, the nested dissection procedure is followed as defined in Section 5.6.

We test against MeTiS' MDO and nested dissection implementation. The default behavior of MeTiS is to use its built in MDO implementation when the input graph is under 100 nodes. This behavior is changed so that it always uses nested dissection when asked to.

Energy - fill-in Another way to measure the performance is by considering the relationship between energy and fill-in. Whereas measuring just the fill-in tells us how useful the QUBO is in practice, measuring the energy - fill-in relationship can tell us how well the QUBO is formulated. The goal of an annealer is to find the lowest energy state of the given QUBO problem. Thus, if the QUBO is well formulated we expect a direct correlation between the energy of the solution and the produced fill-in.

We design an experiment to test this across various graph sizes and graph types. For every QUBO solved in the nested dissection process we usually take the lowest energy solution as the accepted one and continue. In this experiment we are interested in what happens at higher energy. Therefore, we take the 100 lowest energy solutions of each QUBO instead, and we apply the algorithm recursively for each dissection occurring for each of these 100 solutions. This is where the tree representation of solutions is very useful, with each unique order being its own branch of the tree. To accumulate the energies of the multiple dissection steps, the energies are added.

After walking the resulting solution tree, we are left with a collection of solutions each with a unique energy and fill-in pair. Larger graph sizes are associated with higher solution energy, even if it is solved optimally. As such we require a normalization step, where within each graph size the energy and fill-in are normalized between zero and one. This allows us to effectively compare the data between graph sizes. Once the data is normalized every unique energy, fill-in pair is plotted on a scatter plot, and we attempt a linear regression to examine the correlation between energy and fill-in.

6.3.2. Scaling

One of the most limiting factors in quantum computer experimentation is the relatively small size of available hardware. This is also the case for quantum annealers. Therefore an important metric is how the model scales, i.e. how many variables (and by extension qubits) does the model require for a given problem size. As discussed in Section 5.3, domain wall requires less binary variables to represent the same problem space. In one hot encoding, we need m binary variables to represent m states in our logical variable. For domain wall we only require $m - 1$. While this might seem like a marginal improvement, for our case of $m = 3$ this is still a 33% improvement.

Another claimed advantage of domain wall is that the encoding embeds more easily onto the qubit topology. This claim is substantiated by the improved chain breaking rate when compared to one hot encoding. We can reason about this theoretically by recognizing that the domain wall encoding constraint only interacts with neighboring variables. This means at least this constraint can be minor embedded without the need for any chains. However, this does not hold for all constraints. Still, we might expect that because of the different connections between binary bits, this also leads to different (and possibly easier) embeddings.

To test this the following experiment is conducted. A grid graph is generated of varying sizes, ranging from 3x3 to 10x10. A QUBO is generated with both a one-hot and domain wall encoding. Using DWave's minor embedding tool this QUBO problem is minor embedded onto the qubit topology of the Advantage 4.1 system, using the default parameters. At the time of writing this is DWave's most advanced quantum annealer. We then plot the number of qubits used in both embeddings.

6.3.3. Hyperparameter exploration

As mentioned in Section 5.4, hyperparameter optimization is an important but difficult part of proper QUBO formulation. For the presented QUBO it is not feasible to go through the entire hyperparameter search space, since we have five hyperparameters. However, we can still get some intuition for how hyperparameters affect QUBO performance with some simple experiments.

The consequence of badly chosen hyperparameters is at best a poor solution, and at worst no solution at all. Through manual experimentation it is found that there is a high chance a random choice of hyperparameters results in the QUBO being unable to complete the nested dissection process. The chance of successful completion of the algorithm decreases as the problem size increases. This is in part due to the added difficulty of optimally solving large QUBOs, but as is shown later, also due to the choice of hyperparameters. The choice of hyperparameters turns out to be a considerable problem when completing the performance experiments described in the previous section.

With this in mind, in the rest of this section we present several experiments exploring the hyperparameter search space with particular focus on achieving a complete solution.

Size exploration The largest issue when running the performance experiments is the incomplete solutions on larger sized graphs. An incomplete solution manifests when a constraint is broken. Below we summarize the symptoms of a broken constraint.

- **Encoding constraint** - If this constraint is broken the solution no longer makes logical sense. It's impossible to reason about the performance of the algorithm when the encoding is broken.
- **Balance** - While in theory it is possible to have unbalanced results, in practice it is observed that when there is unbalance all nodes are put into one set. This means no dissection is being performed and the algorithm infinitely recurses.
- **Set independence** - In principle this constraint is also not required to produce results we can do fill-in analysis on. However, the QUBO construction code makes the assumption that each dissection step produces independent subgraphs. If a node fails this assumption, it will participate in both of the following dissection steps, meaning it will be present in the elimination list at least twice. To avoid having to cover this case in the code, it is treated as a fatal error.

We assume the sampler is able to find a solution that is acceptably close to the global optimum. There is no way to reliably verify this assumption for an arbitrary input. However, with this assumption we are able to reason that an incomplete solution means the hyperparameters are not optimally tuned. We shall see that this assumption holds insofar we are able to find a hyperparameter configuration that results in complete solutions at larger graph sizes.

In this experiment we are not interested in the performance of the model, as such we don't need to tune the parameters of the goals. This reduces the size of the exploration space considerably. We can reduce the search space even further by coupling the hyperparameters together by means of a multiplier. Failure to solve a graph at larger sizes can manifest for two reasons. Firstly, there might be some implicit degree or graph size dependence in the QUBO (we shall discuss in the following section that this is indeed the case). Secondly, a larger graph means more variables in the QUBO, meaning the linear solver has a larger landscape to explore. For both these cases, scaling the hyperparameters in lock step with a multiplier can uncover if these hypotheses are correct.

The experiment is defined as follows. For each run we multiply the size of the graph with a multiplier. This value is passed as the value of the hyperparameter for every constraint mentioned above. We then run our nested dissection implementation, which returns an ordering if it is successful, and nothing if it is not. We take 100 samples for every multiplier in every dataset. Every dot in the plot shows the percentage of samples that completed successfully. A fully red dot means none succeeded, a fully yellow dot means everything succeeded. The experiments were performed using simulated annealing.

Energy - fill-in As mentioned in a previous section, the energy - fill-in relation is a useful tool for determining how well the QUBO is formulated. In addition to graph size, we can also examine the formulation quality across different hyperparameter configurations. To perform this for the entire search space is infeasible, so as in the previous section we limit ourselves to working with a multiplier. The experiment is very similar to the one defined in Section 6.3.1, except we do not vary the graph size, but instead we vary the hyperparameter multiplier. The graph size is kept constant at 100.

6.4. Performance results

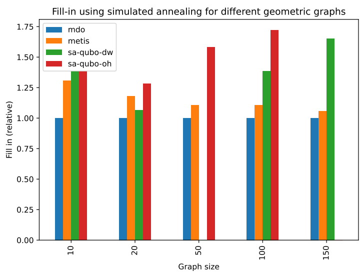
In this section we present the results of the experiments defined in Section 6.3.1.

6.4.1. Fill-in performance

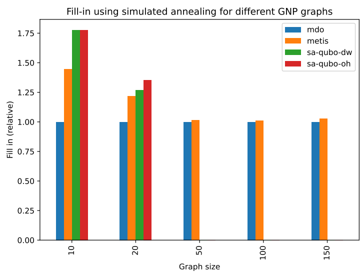
Experiment	**Execution time (s)**
MDO	0.00051
MeTiS	0.00040
QA-QUBO-DW	760
QA-QUBO-OH	560
SA-QUBO-DW	14
SA-QUBO-OH	11

Table 6.1: Execution times of the various ordering methods for a 8x8 grid graphs

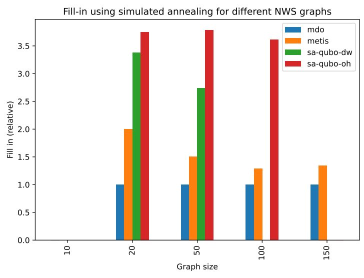
Figure 6.4 shows the performance of the various algorithms on every synthetic graph type except on grid graphs. We can see that for every graph type except for wheel graphs the classical algorithms



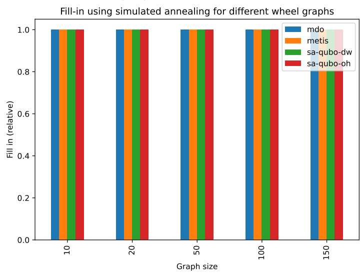
(a) Geometric graph



(b) Erdős-Rényi graph



(c) Newman Watts Strogatz graph



(d) Wheel graph

Figure 6.4: Fill-in produced by the QUBO formulation using simulated annealing on various graph types and sizes. The fill-in value is presented relative to MDO. A lower value is better. The missing bars represent failed experiments.

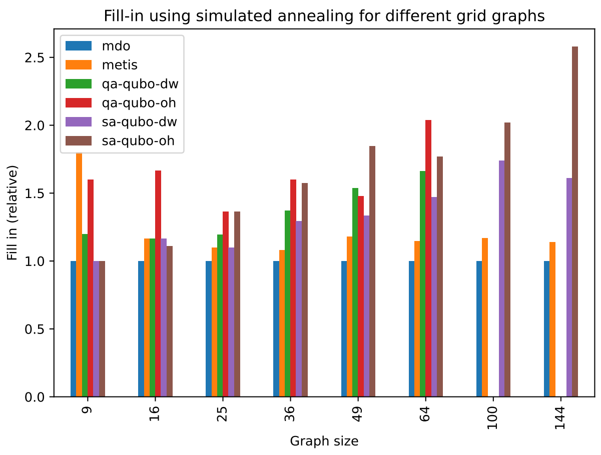
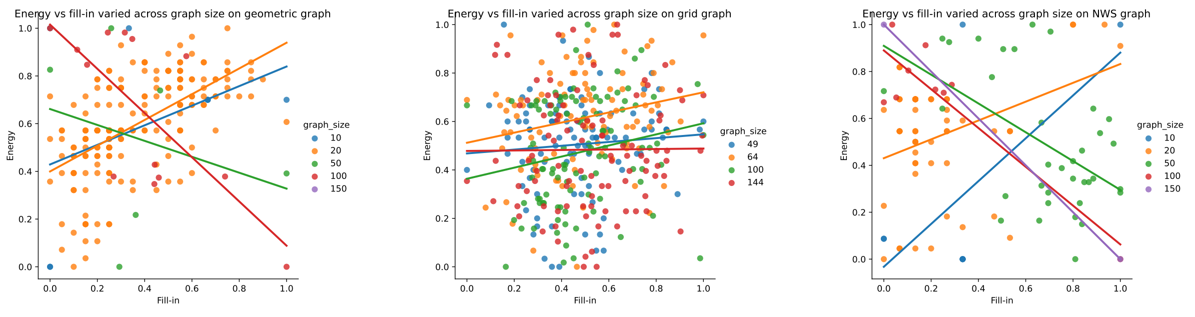


Figure 6.5: Fill-in performance on differently sized grid graphs. Inlcudes results from quantum annealer.



(a) Geometric graph (b) Grid graph (c) Newman Watts Strogatz graph

Figure 6.6: Results of the energy vs fill-in experiment varied over graph sizes. The wheel graph types is omitted due to lack of spread in the data. The Erdős-Rényi graph is omitted because of a lack of solutions at higher graph sizes. The lines represent a linear best fit, for which we expect positive correlation.

outperform the QUBO based nested dissection. The exact performance differs per graph type. The difference between classical approaches and the QUBO is the least pronounced for geometric graphs. However, on NWS graphs the algorithm has trouble finding complete results, and on Erdős-Rényi graphs we see that above 20 nodes the QUBO never finds a complete nested dissection.

The grid graphs are evaluated separately since they include results from the quantum annealer, an experiment that wasn't performed for the other graph types. These results are presented in Figure 6.5. We see that on smaller graph types the performance is quite similar across all methods tried. However, this trend lessens as the graph size increases, and we see that the classical methods outperform the quantum methods. It is interesting to note that on the largest graph size it seems the domain wall encoding performs better than one hot encoding. Furthermore, we are unable to make a decisive conclusion about the comparison between simulated and quantum annealing. However, from the data that is in the figure, it seems simulated annealing performs slightly better than quantum annealing.

The execution times of the various ordering methods for an 8x8 grid graph is shown in Table 6.1. We can see that the execution time of QA is 6 orders of magnitude larger than for MDO and MeTiS. It is is also slower than SA, but just by a single order of magnitude.

6.4.2. Energy - fill-in

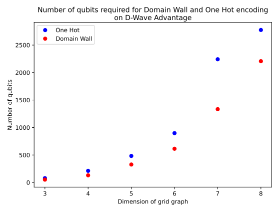
In Figure 6.6 the fill in of solutions with varying energies is plotted. For a well formulated QUBO we expect that the fill-in is positively correlated (i.e. at higher energy we expect a high fill-in). Again the data differs significantly between graph type. The grid graph (Figure 6.6b has a positive correlation for every graph size. This is not the case for the NWS and geometric graph, where for larger graph sizes the corelation becomes negative. We can use this result to explain the poor performance of the QUBO formulation on those respective graphs. The fact that the quality of the formulation differs between graph types suggests that there is a significant deviation in the optimal hyperparameter configuration between different graph types.

6.5. Scaling results

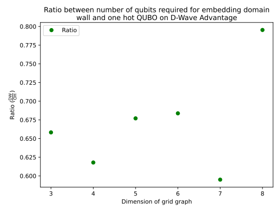
In this section we present the results of the experiments defined in Section 6.3.2. Figure 6.7a shows the result of the minor embedding experiment. We expect there to be at least a $\frac{2}{3}$ ratio between the amount of qubits used by domain wall and one hot, since in our case domain wall only needs two thirds of the variables one hot requires. A ratio between $\frac{2}{3}$ means that domain wall embeds easier than the equivalent one hot QUBO. The turning point is shown on the chart as the orange dotted line.

The results show that domain wall doesn't necessarily embed better than one hot. However, this is not a conclusive experiment. The QUBO model presented in this work is one that is generally difficult to embed due to the fact it has an all-to-all relation in the balance constraint. This constraint has the same connectivity in both one hot and domain wall, and it generally dominates the overall connectivity of the QUBO.

Figure 6.7a does highlight an important advantage of domain wall, which is that it uses less variables by definition. The consequence of this is that a 10x10 grid can still be embedded on Advantage 4.1 by

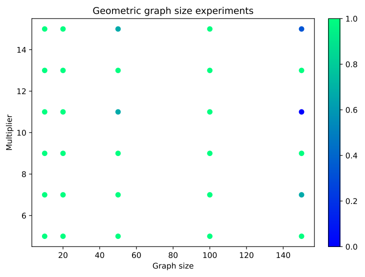


(a) Number of qubits required for different sized grid graphs using both domain wall and one hot encoding.

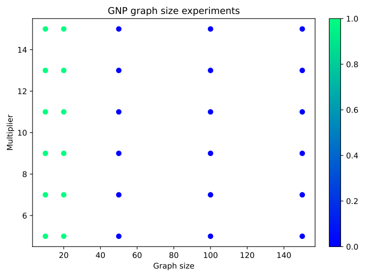


(b) Ratio between qubits required for different sized grid graphs using both domain wall and one hot encoding.

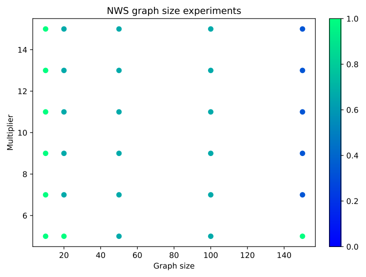
Figure 6.7: Results of the scaling experiments.



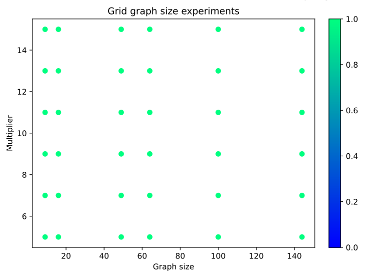
(a) Geometric graph



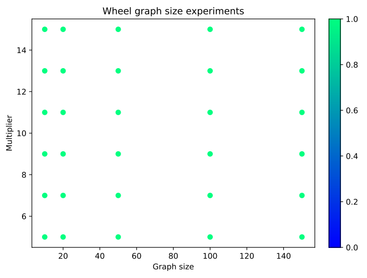
(b) Erdős-Rényi graph



(c) Newman Watts Strogatz graph



(d) Grid graph



(e) Wheel graph

Figure 6.8: Results of the size exploration on various synthetic datasets. Each dot is a particular combination of multiplier and graph size. The color gradient represents the fraction of experiments that succeeded. A dot having a value of 1.0 means all experiments completed successfully.

domain wall, but not by one hot.

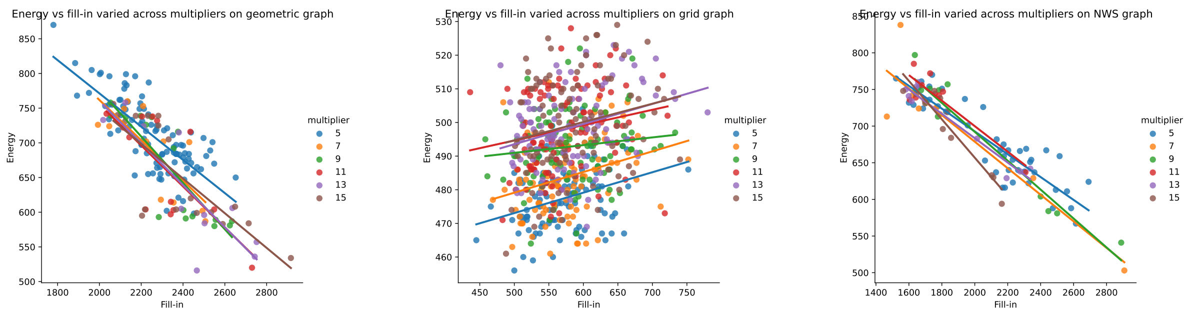
6.6. Hyperparameter exploration results

In this section we present the results of the experiments defined in Section 6.3.3.

6.6.1. Size exploration

Figure 6.8 shows the results for every dataset. The deterministic graph types (Figure 6.8d and Figure 6.8e) all succeeded consistently. While it's expected that there will be less variance for these graph types, there is also some randomness induced by the Monte Carlo process powering simulated annealing. The results of these simulations allowed us to perform every other experiment with confidence they would actually succeed. However, the little change in results show that the multiplier abstraction might be limiting the hyperparameter configurations we are able to explore. If we were truly exploring a comprehensive part of the search space, we would expect some of our configurations to lead to failure.

Despite this, there are some interesting results on the random graph types. These are shown in Figures 6.8a, 6.8b and 6.8c. Both the NWS and Geometric graphs are relatively stable, with most instances being solved 100% of the time. However, the Erdős-Rényi graph does not get solved for any multiplier at any size above or equal to 50 nodes. This is because the graph density ($\frac{\text{number of edges}}{\text{number of nodes}} = \frac{p(n-1)}{2}$. Since the algorithm aims to split the graph into two connected components, at some graph density it no longer makes sense to partition the graph. The explanation for these results is then that this point lies somewhere around the 50 node mark for Erdős-Rényi graphs.



(a) Geometric graph (b) Grid graph (c) Newman Watts Strogatz graph

Figure 6.9: Results of the energy vs fill-in experiment varied over various multipliers. The wheel graph types is omitted due to lack of spread in the data.

These figures were made after finding a dependence on the node degree in the hyperparameter configuration in the domain wall formulation. These findings are described in Section 6.6.3. Without this fix, the results looked considerably worse, with most graph types failing at almost every multiplier. In Section 6.3.3 we made the assumption that the used sampler is able to find a solution acceptably close to the global optimum. This assumption is required to make any conclusions about the configuration of hyperparameters. Since we are able to confirm a degree dependence using this experiment, we can confirm this assumption to the extent that the QUBO solver (in this case simulated annealing) is adequate for us to find this dependence. Consider that the solver could have been so poor it would have been impossible to find any discernible relationships.

6.6.2. Energy - fill-in

In the energy - fill-in experiment with a varying multiplier we see a similar result to the one we observed in the previous section. We see in Figure 6.9 that changing the multiplier has limited effect on the quality of our QUBO formulation. Indeed the corelation between energy and fill-in does change with varying multiplier. However, as an example, the geometric graph stays negatively correlated regardless of which multiplier is chosen. This can suggest we didn't examine a range of multiplier large enough, or that a multiplier simply does not have a large effect on the quality of the formulation.

6.6.3. Domain wall

The use of domain wall encoding warranted a more thorough look at the energetic effects of our goals and constraints. Specifically, as a consequence of hyperparameter exploration experiments it turned out to be important to examine the consequences of the encoding constraint being broken. A break in the encoding constraint caused a large negative spike in energy. For valid solutions the energy is centered around 0, with extra energy for the penalty on nodes in S and edge cuts. However, for solutions with nodes that broke domain wall encoding, the energy was far *below* 0. This shows that the problem is not an insufficient search of the solution space, but rather that it is energetically favorable to break the encoding constraint.

For this particular case, several nodes had broken their domain wall encoding, having a binary value of $[1, 0]$. This led to some sanity checking, and analysis of the energetic effects of each of the constraints. Firstly, it is possible our encoding constraint is not formulated properly, which could explain the tendency to break the encoding. However, $H_{BDW} = 4\kappa$ with $b_0 = 1, b_1 = 0$, which is a positive number and hence cannot be the source of our large negative component.

The rest of the constraints can be considered using a process of elimination. Looking at Table 5.4, it is immediately clear that the only expression able to evaluate to a negative number is the set independence constraint, considering every $b_i \in \{0, 1\}$. Indeed, for $i = [1, 0]$ and $j = [1, 0]$ or $j = [0, 0]$, we find $H_{SI} = -\kappa_{SI}$ or $H_{SI} = -2\kappa_{SI}$. This means that if κ_{SI} is large enough, it can become energetically favorable to break the encoding constraint. Crucially, H_{SI} is negative for $j = [0, 0]$, which means that the energetic effects of a bad encoding are not limited to the invalid nodes only. A single invalid node can potentially introduce multiple instances of negative H_{SI}.

The question then arises, what should κ_{SI} be so that we don't break the encoding, but still ensure our dissection actually produces independent sets? We can consider that κ_{SI} should have a value such that an encoding violation does not subtract more energy than H_{BDW} adds. Every badly encoded node adds $4\kappa_{BDW}$ to the solution energy. Suppose $i = [1,0]$, and every other node $j = [1,0]$ or $j = [0,0]$, then H_{SI} subtracts at least $2\kappa_{SI}d_i$ from the solution energy, where d_i is the degree of node i. This highlights that the effects of this phenomenon depend on the structure of the problem graph. Hence, the correct κ_{SI} changes according to the input. However, we can change the formulation of H_{SI} to scale each term by $\frac{1}{d_i}$, negating the degree dependence. Now, every invalid node $i = [1,0]$ subtracts at least $-2\kappa_{SI}$ from the solution energy, a constant. This allows us to define $2\kappa_{BDW}$ as an upper bound for κ_{SI}.

Similarly, we can find a lower bound for κ_{SI} by considering how the set independence constraint is broken. The biggest factor involved in breaking this constraint is the separator penalty. It incentivizes the solver to look for solutions with the least number of nodes in the separator set. If κ_{SI} is too low, the solver may decide that lowering the number of separator nodes is more energetically favorable than upholding the set independence constraint. Every node that is assigned to the wrong set (i.e. set A or B instead of S) adds $2\kappa_{SI}$ to the solution energy. For every allocated separator the penalty adds κ_S to the solution energy. Hence, the lower bound for κ_{SI} is at least $\frac{1}{2}\kappa_S$.

On computing the degree In the related works we have outlined the minimum degree algorithm, which relies on knowing the degree at every step of the elimination process. This induces considerable computational overhead, causing development of the approximate algorithms that avoid directly calculating node degree as much as possible. To compensate for the degree dependency of the set independence constraint, we propose to divide κ_{SI} by the real degree. This requires knowledge of the degree of every node at every step of the nested dissection process.

Given entire research directions exist to avoid exactly this calculation, how justified is then to explicitly include this calculation in our approach? The complexity of computing the degree in a graph is $O(|E|)$. In minimum degree we must compute the degree every round of elimination, which happens n times. Hence minimum degree has a complexity of $O(n|E|)$. In the case of our QUBO formulation, we do not have to construct the formulation n times. Assuming every dissection roughly halves the graph at every step, we expect $\log n$ dissection steps. Hence we only need to compute the degree $O(|E| \log n)$ times, a better result than minimum degree.

7 Discussion

The main contribution of this work is the formulation of and experimentation with a QUBO problem capable of performing the necessary graph partitioning suitable for nested dissection. A general formulation method is given, able to produce QUBO problems on arbitrary graphs. These QUBO problems were then solved using both a quantum and classical annealing method. We successfully solved several problem instances on the D-Wave quantum annealer. The solution to these problems are used in a matrix reordering heuristic, with the aim of reducing the fill-in during Gaussian elimination. The performance of the QUBO problems are evaluated by considering the fill-in resulting from eliminating the corresponding matrix using the annealing powered ordering. These results are compared to state of the art classical alternatives.

For almost every problem instance the QUBO formulation is outperformed by the classical state of the art. On some small problem instances (3x3 and 4x4 grid graphs) the QUBO formulation solved using simulated annealing outperformed MeTiS. However, the total execution time of the QUBO formulated solutions are several orders of magnitude higher ($\sim 10^6$ times) higher than that of the classical alternatives. This means that as of the moment of writing, performing nested dissection using QUBO formulations for graph partitioning does not offer a compelling alternative to the current state of the art.

There are several reasons for this. The fact that the QUBO formulation produces larger fill-in compared to classical alternatives already shows this method is unsuitable right now. However, even if the QUBO based ordering produced less fill-in, the classical alternatives would still be more attractive due to the large execution time associated with quantum annealing. The long execution time comes primarily from the embedding algorithm. For example, embedding an 8x8 grid takes roughly 10 minutes to complete. Due to the physical limits of quantum annealers this is a required step. Without improvements in embedding time, quantum annealing for nested dissection may remain unattractive, as the embedding time may very well negate any reduction in elimination time.

Besides the novel QUBO formulation, this work also provides insights into the process of designing QUBO problems in an applied setting. During the design process it became clear that the hyperparameters play a significant role in the performance of the formulation. This claim is substantiated by the energy-performance analysis, which suggests that in its current formulation the QUBO problem is not well tuned for effectively solving nested dissection. The large number of goals and constraints also made exploration of the hyperparameter space non-trivial. Exploration was attempted using a method based on multipliers, which fixes the ratio between different parameters, shrinking the search space considerably. Through this exploration, a pattern in the domain wall encoding was identified. Here, another contribution is made, namely the discovery that a violation of domain wall constraints does not merely break the formulations logical interpretation, but also has mathematical consequences. This property was used to derive a limit on some of the hyperparameters, a result which is not trivial when using one-hot encoding.

7.1. Encoding

One of the contributions of this work is a direct applied comparison between the two most popular methods of QUBO encoding. Recall that the goal of the encoding method is to be able to efficiently express a discrete integer problem encoded as a set of binary variables. The constraining factor here is not necessarily the number of binary variables but also the ways by which these variables interact with each other. One can accomplish this with the two most popular methods: domain wall encoding and one hot encoding.

One hot encoding is an intuitive encoding, whereby each state of the integer variable is encoded as a separate binary variable. This means to express an integer problem with m different states, we require m bits.

In domain wall encoding the integer value is encoded in the position of a domain wall within a series of bits. A domain wall is the boundary at which the series has a differing value. So for example, in the bit string "0011" the domain wall is on the second position (regardless of which direction we read). This formulation allows us to express an integer problem using $m - 1$ bits, where m is the number of states we need to represent. It is believed that a domain wall encoding of a problem generally performs better than the one hot equivalent [8]. We investigated this claim by running every experiment for one hot and domain wall.

In general domain wall embedded more easily, primarily due to the lower amount of variables required. Domain wall should also be easier to embed because of a more favorable interaction graph. However, this favorable interaction graph is only for the encoding constraint, and not necessarily for any other goals and constraints. Therefore, while the domain wall encoding constraint is easier to embed, a QUBO problem with more complex goals and constraints does not benefit from this advantage. We conclude that for the embedding performance the primary gain in domain wall for applied QUBO formulations is the reduction of variables.

In addition to evaluating the perform claims, while experimenting with hyperparameter exploration, we found that domain wall has an interesting property when it comes to the enforcement of constraints. This property actually allows one to find mathematical relationships between the different hyperparameters, which is not possible in one hot encoding. Using this property allowed us to form a formulation that was adequate enough to run all the displayed performance experiments.

7.1.1. Performance

In the original paper introducing the domain wall encoding the author mentions several advantages of using domain wall over one hot encoding [8]. For example, in relation to quantum annealers, the author mentions that the domain wall encoding does not need to go through an invalid state to arrive at a new one. In other words, we only need a single bit flip to arrive at a new valid state. This is in contrast to one hot, which requires at least two. In theory this leads to a better performing QUBO.

We attempted to evaluate this theory by considering both a one hot and domain wall formulation in our performance experiments. In general, it seems the domain wall formulation performed better. In all but two experiments when comparing the relative performance of domain wall and one hot, the domain wall encoded QUBO produced an ordering that results in lower fill-in. The two experiments in which this was not the case were on the grid graph instance at 4x4 and 7x7 instance. Given that annealing and embedding is a stochastic process, and that on the grid graphs the experiments were not repeated, these cases are treated as an outlier. The overall better performance of domain wall is also confirmed by some literature [9].

However, it is difficult to attribute this advantage to a particular cause. Apart from having a lower barrier to state transition, the domain wall encoding also enjoys the use of less qubits overall. A QUBO with less variables is easier to solve by definition, which means that the improvements we observed in the performance results may well be due to the fact that domain wall encoded QUBOs have less variables. Since the variable number advantage is one of constant offset ($m - 1$ for domain wall and m for one hot), one could make a more informed conclusion about the reason domain wall performs better by considering problems where $m >> 1$, making the effect of lower number of variables negligible. Given that this work is primarily focused on the performance evaluation of QUBO formulations for nested dissection (which has $m = 3$), this investigation falls out of scope. However, as future work it is interesting to consider the performance of the encoding schemes on a QUBO with many discrete options.

Other than the relative intuition of one hot, we see no reason not to express every QUBO using domain wall.

7.1.2. Violations

One of the most interesting aspects when comparing one hot and domain wall encoding is the effects of encoding violations in each case. In one hot encoding, an encoding violation leads to a logical error. While the result doesn't necessarily have to make sense, an invalid one hot variable still lives in the realm of the specified constraints and goals. For example, suppose we take our dissection QUBO, and consider a node that violates the one hot encoding. We now have a node that is either assigned to multiple sets or is not assigned to any. Obviously in the context of our problem, both these cases are meaningless. It is not satisfactory to simply not order a node, and neither is it appropriate to order a node twice. Nevertheless, in both these cases the allocated nodes still respect our constraints. If a node is placed in two sets, it is still properly considered in the balance constraint. It will also still be considered in the set independence constraint, and penalized if any other unfortunate nodes happen to be connected to it. While the solution may be contextually meaningless to the user, mathematically the model still produces sensible results. This is not the case for a domain wall encoded model. In a domain wall encoded model, an encoding error instantly renders the model both logically and mathematically useless.

At first glance this property of domain wall may seem like a disadvantage. However, Section 6.6.3 demonstrates how we can use this phenomenon to our advantage. In general, as discussed in Section 5.4, it is difficult to systematically determine which hyper-parameters constitute an optimal QUBO. The mathematical breakdown of the domain wall encoding on encoding violations at least in some cases is able to provide a way to derive a concrete relationship between hyper-parameters. For example, in the dissection QUBO it is possible to identify how certain parameters scale and, crucially, how they scale in relation to each other. This reduce the dimensions of the hyper-parameter search space.

It is not guaranteed that these opportunities arise in every domain wall encoded problem. Nevertheless, it is still important to highlight the possibility that we domain wall can provide clear relationships between hyper-parameters.

7.2. Performance

As mentioned previously, the performance compared to classical methods is subpar. In general the QUBO nested dissection produces worse results when compared to MeTiS, and requiring substantially more time to arrive at this result. As such, it is not recommended to use this method over the available classical methods.

Nonetheless, this does not allow us to discard the notion of quantum integration in linear solver workflows altogether. We currently do not use simulated annealing to solve the graph partitioning portion of nested dissection due to its poor performance when compared with other heuristics. This poor performance is demonstrated in performance results. Notably, simulated annealing also required significantly more compute time to arrive at a worse result.

However, this does not say anything about the possible future for quantum annealing. In theory, quantum annealing arrives to a global optimum asymptotically quicker than simulated annealing does, under adiabatic conditions. Notably, in our results the quantum annealer performed on par with the simulated annealer, while the annealing time for the quantum solver was orders of magnitude lower. There is currently still significant setup time associated with the quantum annealer, which means that the overall experiment time was higher for the quantum annealer. However, the *actual* annealing time is orders of magnitude lower. This is an important result, as it shows the power of quantum annealing compared to simulated annealing, even in the beginning stages of the technology. Therefore, it is possible that with more technological advancement it is the case that this QUBO formulation can outperform the classical approaches in both fill-in and overall compute time.

To reach this point there are still several advancements that need to be made. We will consider the requirement of advancements in hyperparameter exploration in a following section. Besides that, we note that currently a quantum annealing problem endures extremely high setup time due to the relative difficulty of minor embedding. To be able to confidently assert the advantage of quantum annealing, it is important that the cost of minor embedding is reduced. It is a difficult issue, since it is itself an NP-Complete problem. Nevertheless, as long as the cost of minor embedding is larger than the cost of directly computing the nested dissection, quantum annealing does not offer any advantage over classical methods.

7.3. Methods

Apart from the limitation of quantum annealing, there are also several ways we can improve the proposed method.

7.3.1. Graph coarsening

The state of the art nested dissection package MeTiS is designed to work on large sparse input graphs. However, it does not perform its partitioning heuristic on graphs of such scale. Instead it uses graph coarsening techniques to reduce the problem size due to around 100 vertices, and performing partitioning on that. It is possible to use a similar approach to reduce the size of the QUBO problems we have to evaluate. As is shown in the results, the QUBO formulation performs drastically worse for larger input sizes, meaning we can heavily benefit from smaller graphs. Furthermore, we can also use this to potentially overcome the limited physical size of current quantum annealers. A potential future avenue of research is investigating the effect of coarsening graphs on the utility of the QUBO formulation.

7.3.2. Hyperparameters

One of the biggest difficulties in developing the QUBO formulation was the impact of hyperparameters on the performance of the formulation. In fact, the initial mathematical formulation was relatively easy compared to the task of picking proper hyperparameters. The suboptimal pick of hyperparameters also prohibited proper evaluation of the model's performance, as the approach was only able to evaluate small (and trivial) graphs. What we have decidedly observed from the various energy - fill-in experiments is that both the current hyperparameter configuration, and the methods to explore the possible configurations, is inadequate. We see weak correlation between energy and fill-in at best, with several graph types even having a negative corelation. The model has the correct goals and constraints, evidenced by the production of valid nested dissection orderings. Assuming we are not limited by the underlying solver, it leaves no other possibility that the model is badly tuned. We expect that there is a set of hyperparameters for which the correlation between energy and fill-in is a positive one. We also expect this would improve the performance of the model.

We have attempted to explore the search space of the hyperparameters using a multiplier analogue. This multiplier ties the value of three different hyperparameters together, such that a three dimensional search space is now a one dimensional one. It turns out that this approach of exploring the hyperparameter search space is inadequate. This is evidenced by the fact that no change in multiplier significantly changes the corelation between energy and fill-in. Given that this is exactly the metric that we are trying to improve, this is not satisfactory.

Hence an alternative way to explore the hyperparameter search space is required. Currently, this is an open research problem. Naively exploring a large search space becomes computationally infeasible for more complex QUBO problems like the one presented in this work. That might also not lead to a universal solution, as we have seen that the quality of the QUBO formulation doesn't just depend on a specific set of hyperparameters, but also on the graph type and graph size. This does imply that patterns can be identified on the basis of the input problem. Indeed, we have been able to identify one hyperparameter pattern in this work. However, we are not aware of a mathematical framework by which we can accomplish this in a structured manner. Perhaps this can be developed with enough research.

Thus we conclude that in the context of this work, the lack of a feasible hyperparameter exploration strategy hampered the performance of the formulation. We expect that if more reliable methods and computationally feasible methods for hyperparameter exploration are available, we are able to find a set of hyperparameters for which the model performs better. Looking beyond this work, we can also state that the lack of proper techniques for hyperparameter exploration pose a large threat to the adoption of quantum annealing techniques or metaheuristic techniques in general.

7.4. Research goals

At the beginning of this work we stated several research questions that drove the direction for this work. For clarity, they are repeated here.

- How can we formulate the nested dissection graph partitioning problem as an optimization problem suited for quantum annealing?

- How well does this formulation perform when compared to other heuristic methods for performing nested dissection?
- What does the inclusion of a quantum step mean for the Finite Element Analysis pipeline?

In principle the first question has been confidently answered by the expression of the minimum vertex separator problem as a QUBO using multiple encoding techniques. However, what remains unclear is if this is the most optimal formulation of the problem. There is still more clarity needed on the topic of hyperparameters, a vital part of the QUBO formulation.

This is also related to the response to the second question. With the current formulation the QUBO formulation performs rather poorly when compared to other heuristic methods for performing nested dissection. This is if one considers the performance both in the produced fill-in, and the time it took to arrive to that solution. Again, the importance of hyperparameters is emphasized. They most likely have a large impact on the performance of the formulation, and as such, without more research into the hyperparameter configuration, it is unclear if the performance is yet to be improved, even on current hardware.

Lastly, in this work we also presented a computational model for Finite Element Analysis that includes a quantum step. In principle not much changes, except there is some more preprocessing involved in formulating the QUBO. As of right now, the inclusion of a quantum step also implies the need for cloud access, as there are no commodity quantum annealers available that a downstream user could purchase. However, there is no guarantee that with more technological advancement that will always stay the case.

Bibliography

[1] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. “An approximate minimum degree ordering algorithm”. In: *SIAM Journal on Matrix Analysis and Applications* 17.4 (1996), pp. 886–905.

[2] C Ashcraft et al. “Nested dissection revisited”. In: (2016). URL: `https://epubs.stfc.ac.uk/manifestation/27193200/RAL-TR-2016-004.pdf`.

[3] Egon Balas and Cid C. De Souza. “The vertex separator problem: A polyhedral investigation”. In: *Mathematical Programming* 103.3 (July 2005), pp. 583–608. ISSN: 00255610. DOI: `10.1007/S10107-005-0574-7`.

[4] Piotr Bermanf and Georg Schnitger. “ON THE PERFORMANCE OF THE MINIMUM DEGREE ORDERING FOR GAUSSIAN ELIMINATION*”. In: *SIAM J. MATRIX ANAL. APPL* 11.1 (1990), pp. 83–88. URL: `https://epubs.siam.org/terms-privacy`.

[5] Dominic W. Berry, Andrew M. Childs, and Robin Kothari. “Hamiltonian Simulation with Nearly Optimal Dependence on all Parameters”. In: *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS* 2015-December (2015), pp. 792–809. ISSN: 02725428. DOI: `10.1109/FOCS.2015.54`. arXiv: `1501.01715`.

[6] Zhengbing Bian et al. *Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results*. 2020. DOI: `10.1016/j.ic.2020.104609`. URL: `https://doi.org/10.1016/j.ic.2020.104609`.

[7] Carlos Bravo-Prieto et al. “Variational Quantum Linear Solver”. In: *Quantum* 7 (2023). ISSN: 2521327X. DOI: `10.22331/q-2023-11-22-1188`. arXiv: `1909.05820`.

[8] Nicholas Chancellor. “Domain wall encoding of discrete variables for quantum annealing and QAOA”. In: *Quantum Science and Technology* 4.4 (2019). ISSN: 20589565. DOI: `10.1088/2058-9565/ab33c2`. arXiv: `1903.05068`.

[9] Jie Chen, Tobias Stollenwerk, and Nicholas Chancellor. “Performance of Domain-Wall Encoding for Quantum Annealing”. In: *IEEE Transactions on Quantum Engineering* 2 (2021), pp. 1–16. ISSN: 26891808. DOI: `10.1109/TQE.2021.3094280`. arXiv: `2102.12224`.

[10] Barry A Cipra. “An Introduction to the Ising Model”. In: *Source: The American Mathematical Monthly* 94.10 (1987), pp. 937–959.

[11] Elizabeth Cuthill and James McKee. “Reducing the bandwidth of sparse symmetric matrices”. In: *Proceedings of the 1969 24th national conference*. 1969, pp. 157–172.

[12] D-Wave. *Minor Embedding*. `https://docs.dwavequantum.com/en/latest/quantum_research/embedding_intro.html`. Accessed: 2024-12-15.

[13] Hristo Djidjev et al. “Efficient Combinatorial Optimization Using Quantum Annealing”. In: (2016). arXiv: `1801.08653v2`.

[14] Bojia Duan et al. “A survey on HHL algorithm: From theory to application in quantum machine learning”. In: *Physics Letters, Section A: General, Atomic and Solid State Physics* 384.24 (2020), p. 126595. ISSN: 03759601. DOI: `10.1016/j.physleta.2020.126595`. URL: `https://doi.org/10.1016/j.physleta.2020.126595`.

[15] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, Jan. 2017. ISBN: 9780198508380. DOI: `10.1093/acprof:oso/9780198508380.001.0001`. URL: `https://doi.org/10.1093/acprof:oso/9780198508380.001.0001`.

[16] Jacek Dziarmaga. “Dynamics of a Quantum Phase Transition: Exact Solution of the Quantum Ising Model”. In: *Phys. Rev. Lett.* 95 (24 Dec. 2005), p. 245701. DOI: `10.1103/PhysRevLett.95.245701`. URL: `https://link.aps.org/doi/10.1103/PhysRevLett.95.245701`.

[17] P Erdos and A Renyi. “On random graphs I”. In: *Publ. math. debrecen* 6.290-297 (1959), p. 18.

[18] Alexandre Ern and Jean-Luc Guermond. “ESAIM: Mathematical Modelling and Numerical Analysis EVALUATION OF THE CONDITION NUMBER IN LINEAR SYSTEMS ARISING IN FINITE ELEMENT APPROXIMATIONS”. In: 40.1 (2006), pp. 29–48. DOI: `10.1051/m2an:2006006`. URL: `www.edpsciences.org/m2an`.

[19] Shimon Even. “ An Algorithm for Determining Whether the Connectivity of a Graph is at Least k ”. In: *SIAM Journal on Computing* 4.3 (1975), pp. 393–396. ISSN: 0097-5397. DOI: `10.1137/0204034`.

[20] Dmitry A. Fedorov et al. “VQE method: a short survey and recent developments”. In: *Materials Theory* 6.1 (2022). DOI: `10.1186/s41313-021-00032-6`. arXiv: `2103.08505`.

[21] A. B. Finnila et al. “Quantum annealing: A new method for minimizing multidimensional functions”. In: *Chemical Physics Letters* 219.5-6 (Mar. 1994), pp. 343–348. ISSN: 0009-2614. DOI: `10.1016/0009-2614(94)00117-0`. arXiv: `9404003 [chem-ph]`.

[22] Stuart Geman and Donald Geman. “Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-6.6 (1984), pp. 721–741. DOI: `10.1109/TPAMI.1984.4767596`.

[23] Alan George. “NESTED DISSECTION OF A REGULAR FINITE ELEMENT MESH*”. In: *SIAM J. NUMER. ANAL* 10.2 (1973). URL: `https://epubs.siam.org/terms-privacy`.

[24] Alan George and Joseph W H Liu. “THE EVOLUTION OF THE MINIMUM DEGREE ORDERING ALGORITHM”. In: 31 (1 1989), pp. 1–19. URL: `https://epubs.siam.org/terms-privacy`.

[25] John R. Gilbert, Donald J. Rose, and Anders Edenbrandt. “A SEPARATOR THEOREM FOR CHORDAL GRAPHS”. In: *Journal of Algorithms* 5.3 (1984), pp. 391–407. ISSN: 01966774. DOI: `10.1016/0196-6774(84)90019-1`. URL: `https://epubs.siam.org/terms-privacy`.

[26] George Karypis and Vipin Kumar. “A fast and high quality multilevel scheme for partitioning irregular graphs”. In: *SIAM Journal of Scientific Computing* 20.1 (1998), pp. 359–392. ISSN: 10648275. DOI: `10.1137/S1064827595287997`.

[27] George Karypis, Kirk Schloegel, and Vipin Kumar. “P AR M E I S □ Parallel Graph Partitioning and Sparse Matrix Ordering”. In: *Memory* 95.January (2003), pp. 1–29.

[28] Sophia Kolak et al. “Evaluating Quantum Algorithms for Linear Solver Workflows”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 13999 LNCS (2023), pp. 634–647. ISSN: 16113349. DOI: `10.1007/978-3-031-40843-4_47/FIGURES/6`. URL: `https://link.springer.com/chapter/10.1007/978-3-031-40843-4_47`.

[29] Tomoko Komiyama and Tomohiro Suzuki. “Sparse Matrix Ordering Method with a Quantum Annealing Approach and its Parameter Tuning”. In: *Proceedings - 2021 IEEE 14th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2021* 1 (2021), pp. 258–264. DOI: `10.1109/MCSoC51149.2021.00045`.

[30] Julia Kwok and Kristen Pudenz. “Graph Coloring with Quantum Annealing”. In: (). arXiv: `2012.04470v1`.

[31] Dominique LaSalle and George Karypis. “Efficient nested dissection for multicore architectures”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9233 (2015), pp. 467–478. ISSN: 16113349. DOI: `10.1007/978-3-662-48096-0_36/TABLES/5`. URL: `https://link.springer.com/chapter/10.1007/978-3-662-48096-0_36`.

[32] Xiaoye S. Li. “An overview of SuperLU: Algorithms, implementation, and user interface”. In: *ACM Transactions on Mathematical Software* 31 (3 2005), pp. 302–325. ISSN: 00983500. DOI: `10.1145/1089014.1089017`.

[33] J Lipton and R E Tatjan. “A Separator Theorem for Graphs of Bounded Genus”. In: *JOURNAL OF ALGORITHMS* 5 (1984), pp. 177–189.

[34] Richard J. Lipton and Robert Endre Tarjan. “A Separator Theorem for Planar Graphs”. In: *https://doi-org.tudelft.idm.oclc.org/10.1137/0136016* 36.2 (July 1979), pp. 177–189. ISSN: 00361399. DOI: `10.1137/0136016`. URL: `https://epubs-siam-org.tudelft.idm.oclc.org/doi/10.1137/0136016`.

[35] Joseph W.H. Liu. “Modification of the minimum-degree algorithm by multiple elimination”. In: *ACM Transactions on Mathematical Software (TOMS)* 11.2 (1985), pp. 141–153. ISSN: 15577295. DOI: `10.1145/214392.214398`.

[36] Aamir Mandviwalla, Keita Ohshiro, and Bo Ji. “Implementing Grover’s Algorithm on the IBM Quantum Computers”. In: *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018* (2018), pp. 2531–2537. DOI: `10.1109/BigData.2018.8622457`.

[37] Harry M Markowitz. “The Elimination form of the Inverse and its Application to Linear Programming”. In: (1957). DOI: `10.1287/mnsc.3.3.255`. URL: `http://pubsonline.informs.orghttp//www.informs.org`.

[38] McGreevy. “Where do quantum field theories come from?” University of California San Diego. Lecture Notes. Phsyics 293a. 2024. URL: `https://mcgreevy.physics.ucsd.edu/s14/239a-lectures.pdf`.

[39] Nicholas Metropolis et al. “Equation of State Calculations by Fast Computing Machines”. In: *The Journal of Chemical Physics* 21.6 (June 1953), pp. 1087–1092. ISSN: 0021-9606. DOI: `10.1063/1.1699114`. URL: `/aip/jcp/article/21/6/1087/202680/Equation-of-State-Calculations-by-Fast-Computing`.

[40] Satoshi Morita and Hidetoshi Nishimori. “Mathematical foundation of quantum annealing”. In: *J. Math. Phys* 49 (2008), p. 125210. DOI: `10.1063/1.2995837`. URL: `https://doi.org/10.1063/1.2995837`.

[41] Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. “Random graphs with arbitrary degree distributions and their applications”. In: *Physical review E* 64.2 (2001), p. 026118.

[42] Shuntaro Okada, Masayuki Ohzeki, and Shinichiro Taguchi. “Efficient partition of integer optimization problems with one-hot encoding”. In: *Scientific reports* 9.1 (2019), p. 13036.

[43] Mathew Penrose. *Random geometric graphs*. Vol. 5. OUP Oxford, 2003.

[44] CY She and H Heffner. “Simultaneous measurement of noncommuting observables”. In: *Physical Review* 152.4 (1966), p. 1103.

[45] Gagandeep Singh. “Short introduction to finite element method”. In: *Norwegian University of Science and Technology* (2009).

[46] Hayato Ushijima-Mwesigwa, Christian F A Negre, and Susan M Mniszewski. “Graph Partitioning using Quantum Annealing on the D-Wave System”. In: 17 (). DOI: `10.1145/3149526.3149531`. URL: `https://doi.org/10.1145/3149526.3149531`.

[47] Guoming Wang. “EFFICIENT QUANTUM ALGORITHMS FOR ANALYZING LARGE SPARSE ELECTRICAL NETWORKS”. In: (2017). arXiv: `1311.1851v10`.

[48] Mihalis Yannakakis. “Computing the minimum fill-in is NP-complete”. In: *SIAM Journal on Algebraic Discrete Methods* 2.1 (1981), pp. 77–79.