Literature Review

Raufs Dunamalijevs

Contents

[1 Introduction 3](#_Toc88185823)

[2 Annealing in Combinatorial Optimisation 4](#_Toc88185824)

[2.1 Annealing 4](#_Toc88185825)

[2.2 Simulated Annealing 4](#_Toc88185826)

[2.3 Quantum Annealing 5](#_Toc88185827)

[2.4 Digital Annealer 6](#_Toc88185828)

[3 QUBO 8](#_Toc88185829)

[3.1 QUBO Model 8](#_Toc88185830)

[3.2 Constraints and Penalties 9](#_Toc88185831)

[3.3 Natural QUBO Formulation 10](#_Toc88185832)

[3.3.1 Minimum Vertex Cover 10](#_Toc88185833)

[3.3.2 Example Solution 11](#_Toc88185834)

[3.4 Non-Natural QUBO Formulation and Slack Variables 12](#_Toc88185835)

[3.4.1 QUBO Formulation 12](#_Toc88185836)

[3.4.2 Slack Variables 12](#_Toc88185837)

[3.5 Algorithms for Solving QUBOs 13](#_Toc88185838)

[3.5.1 Simulated Annealing 13](#_Toc88185839)

[3.5.2 Tabu Search 13](#_Toc88185840)

[3.5.3 Decomposing Solver 13](#_Toc88185841)

[3.5.4 Steepest Descent Algorithm 14](#_Toc88185842)

[4 Penalty Coefficient Optimisation Techniques 15](#_Toc88185843)

[4.1 Analytical 15](#_Toc88185844)

[4.2 Numerical 1 15](#_Toc88185845)

[4.3 Numerical 2 15](#_Toc88185846)

[4.4 Machine Learning 17](#_Toc88185847)

[5 Summary 19](#_Toc88185848)

[References 20](#_Toc88185849)

List of Figures

[Figure 1: Comparison of node transitions in Simulated and Quantum Annealing (Johnson et al., 2011). 6](#_Toc88185850)

[Figure 2: Visual representation of normal subsets and a vertex cover. 10](#_Toc88185851)

[Figure 3: Example MVC graph. 11](#_Toc88185852)

[Figure 4: A solution to the example MVC. 12](#_Toc88185853)

[Figure 5: Solver Surrogate training (Huang et al., 2021). 17](#_Toc88185854)

[Figure 6: Estimating penalty coefficient with Solver Surrogate (Huang et al., 2021). 18](#_Toc88185855)

List of Tables

[Table 1: Few known constraint/penalty pairs (Glover, Kochenberger and Du, 2019). 9](#_Toc88185856)

# 1 Introduction

# 2 Annealing in Combinatorial Optimisation

In this section, we will discuss a class of metaheuristics inspired by physical *annealing*. We will explain what it is and talk about optimisation metaheuristics that are relevant to this work. Finally, we will look at the Digital Annealer – specialised hardware that can efficiently run one of such metaheuristics.

## 2.1 Annealing

Annealing is a heat treatment process of matter like glass or metal (Collins, 1921; Wu and Fan, 2020). It involves heating the material and carefully cooling it down at a controlled rate to achieve a stable crystal structure. If, however, this rate is exceeded, the resulting crystal will have defects as only locally optimal structures will be allowed to form (Kirkpatrick, Gelatt and Vecchi, 1983). The lack of equilibrium will make such crystal metastable.

## 2.2 Simulated Annealing

The simulated annealing (SA) metaheuristic uses the temperature concept to approximate the global optimum of combinatorial optimisation (CO) problem (Kirkpatrick, Gelatt and Vecchi, 1983; Bertsimas and Tsitsiklis, 1993). The temperature (*T*) is a variable that controls the likelihood of making locally non-optimal uphill moves. Just as in physical annealing, we start optimising with a high temperature and decrease it at a controlled rate. If the rate is low enough, we are guaranteed to find a global optimum (Liang, Cheng and Lin, 2014). It is possible to find good approximations of the optima faster at higher cooling rates.

The temperature allows the algorithm to explore the search space to find the optimal solution (Poole and Mackworth, 2017). The SA starts with a high temperature and acts as a random walk, choosing random moves with high probability, which prevents it from getting stuck at local optima. As the temperature decreases, the algorithm prioritises advantageous moves, acting more like a greedy algorithm. As we shift from higher to lower temperatures, we gradually change from exploring the search space to finding the optimum.

Let *c* be the current node, *n* be the next node (neighbour of node *c*) and *evaluation(x)* be the cost function. *ΔE* is an energy transition (negative when moving downhill) that the proposed node will introduce.

*ΔE* and *T* are the variables that affect the probability of choosing node *n*, where *T* controls the influence of *ΔE* (Johnson *et al.*, 1989).

The *acceptance probability function* maps the current node, proposed node, and temperature to a probability. Many functions suit this purpose, the original being exponential (Kirkpatrick, Gelatt and Vecchi, 1983).

If acceptance probability is higher than a uniformly generated random number, node *c* is rejected, favouring node *n*. However, when (downhill movement), we will always choose the proposed node (Kirkpatrick, Gelatt and Vecchi, 1983).

Pseudocode of SA (Johnson *et al.*, 1989) is given by:

c ← initial node

**while** T > threshold

n ← generate next node(c)

ΔE ← evaluate(n) – evaluate(c)

**if** ΔE ≤ 0 t

c ← n

**else if** P(c, n, T) ≥ random(0, 1)

c ← n

**end**

T ← cooling(T)

**end**

## 2.3 Quantum Annealing

The quantum annealing (QA) metaheuristic uses quantum-mechanical fluctuations to find a global minimum of a function (Ohzeki and Nishimori; Tanaka and Tamura, 2012). In QA, we start with a strong quantum field and gradually decrease its strength (quantum fluctuation). It is analogous to SA, where we reduced the temperature (thermal fluctuation) to achieve the same result.

QA begins in a uniform superposition of all possible states, but as the strength of the quantum field decreases, we approach a single solution state (Farhi *et al.*, 2001).

In SA, neighbouring nodes lie one move away from the current node. In QA, the neighbouring nodes are in a particular range defined by the strength of the quantum field, slowly ceasing in the process of annealing. The transition from one state to another can happen within that range. If there is a short wall between two states, the SA algorithm will need to make a series of disadvantageous moves to reach a better solution or, in other words, *“climb”* over the wall. This is avoided in QA as there is a capacity to tunnel through short walls if the quantum field is strong enough, making QA faster than SA when dealing with search spaces with many thin barriers (Crosson and Harrow, 2016). This effect is called *quantum tunnelling*. A comparison of thermal hopping and quantum tunnelling is shown in Figure 1.

![Shape, arrow

Description automatically generated]()

Figure 1: Comparison of node transitions in Simulated and Quantum Annealing (Johnson et al., 2011).

The main benefit of QA is that it rapidly samples a wide range of configurations when exploring the energy landscape (Higham and Bedford, 2021). QA devices have been developed and commercialised by D-wave (Johnson *et al.*, 2011). They are used to solve real-world problems (D-Wave Systems, 2021).

## 2.4 Digital Annealer

Digital Annealer (DA) is quantum-inspired custom CMOS hardware developed by Fujitsu Laboratories (Aramon *et al.*, 2019). Its algorithm is based on SA but has three significant differences:

1. Instead of considering a single neighbouring node and choosing it if the acceptance probability is higher than a uniform random number, DA looks at all neighbouring nodes in parallel. If DA accepted more than one node, it would select one uniformly at random. Such procedure is advantageous as in SA, if the node was rejected, we need to generate and consider a new one, which takes time. In DA, however, all the nodes are considered simultaneously, making the acceptance probability per “turn” higher.
2. DA uses *dynamic offset*: if no nodes have been accepted, it will artificially increase the following acceptance probability to help the algorithm overcome narrow barriers.
3. All runs begin with the same random state to save time, preventing DA from evaluating initial nodes and their neighbours’ multiple times. Parallel SA generates the initial states for each run separately.

DA pseudocode reflects those differences (Aramon *et al.*, 2019):

initial state ← random state

**for** each run

current state ← initial state

Eoffset ← 0

**for** each step

update temperature if needed

**for** each neighbour *(in parallel)*

calculate ∆Eneighbour

consider neighbour using ∆Eneighbour - ∆Eoffset

if neighbour accepted, record

**end**

**if** accepted neighbours ≥ 1

choose one accepted neighbour uniformly at random

update current state and the energy landscape *(in parallel)*

Eoffset ← 0

**else**

increase Eoffset

**end**

**end**

**end**

Ohzeki *et al.* (2019) have found that DA can be faster and produce more precise solutions than D-Wave 2000Q (penultimate generation of D-Wave’s QA devices) in certain scenarios. DA’s are being used in industry and science to solve CO problems (Fujitsu, no date; Snelling *et al.*, 2020)

Currently, QA devices are expensive and challenging to run as the technology is still under development (Şeker *et al.*, 2020). D-Wave devices suffer from different problems, including qubit noise that results in lower precision (Katzgraber and Novotny, 2018). These problems are actively worked on and improved from one generation of devices to another (Dattani, Szalay and Chancellor, 2019). As QA technology develops, it will likely outperform DA by a considerable amount due to its quantum parallelism (Boyd, 2018).

# 3 QUBO

Quadratic Binary Unconstrained Optimisation (QUBO) is a mathematical formulation that can be applied to many CO problems (Kochenberger *et al.*, 2014). Many problems from industry, government, and science can be reformulated in QUBO (Glover, Kochenberger and Du, 2019).

QUBO solvers include D-Wave QA devices, which can also solve equivalent Ising models (Lucas, 2014; Cruz-Santos, Venegas-Andraca and Lanzagorta, 2019), and Fujitsu DA’s (Aramon *et al.*, 2019). Section 2 describes both.

This section will discuss the structure of QUBO, how to reformulate constrained CO problems into unconstrained QUBO models using penalties and slack variables, and algorithms used to solve them.

## 3.1 QUBO Model

The QUBO model has been described in detail by Glover, Kochenberger and Du (2019). It is expressed by equation 1.

Where *y* is the value to be optimised for, *x* is a vector of decision variables, and *Q* is a square matrix with coefficients. QUBO models are unconstrainted; the only restriction is that variables in decision vector *x* should be 0 or 1. All the information needed for the optimisation is inside the *Q* matrix. QUBO problems are NP-hard.

We will use the minimisation problem (3.2) to demonstrate how to convert a Boolean function into QUBO.

Function 3.2 has two parts: linear and quadratic. As *xj* belong to {0, 1}, we can easily make the linear part quadratic t00 (02=0 and 12=1).

We can express (3.3) using matrixes in the form of a QUBO model.

## 3.2 Constraints and Penalties

The previous subsection shows how to reformulate unconstrained minimisation problems into QUBO. Constrained problems can also be reformulated into this model (Glover, Kochenberger and Du, 2019) by expressing the constraints as quadratic penalties and adding them to the original objective function (the function we are optimizing). The larger penalty is imposed when more constraints are broken and when they are important. When no constraint is violated, the penalty is equal to 0. Since we are solving a minimisation problem, solutions that break constraints and impose a penalty on the objective function will be avoided.

*M* is a positive penalty scalar, also called *the penalty coefficient* (Verma and Lewis, 2020). It controls the effect that the broken constraints will have on the objective function. If a constraint is soft and can be broken, *M* may be decreased, which will reduce the impact of the quadratic penalty on the optimized function. It is possible to use multiple scalars for various constraints if they have different importance, but usually, only a single *M* is used (Glover, Kochenberger and Du, 2019).

If the penalty coefficient is too low, the solver will undervalue broken constraints, and the solution produced by the optimiser will be infeasible. On the other hand, if the penalty coefficient is too large, the penalties will overwhelm the objective function making it harder to differentiate between good and bad solutions (Glover, Kochenberger and Du, 2019). It is possible to use domain expertise to select an acceptable *M* (Glover, Kochenberger and Du, 2019), but carefully chosen penalty coefficients can produce solutions of better quality (Şeker *et al.*, 2020). Finding optimal *M* is not trivial and different techniques have been proposed to estimate it (Verma and Lewis, 2020; Huang *et al.*, 2021).

Quadratic penalties for certain constraints are already known. Few are shown in Table 1.

|  |  |
| --- | --- |
| Classical Constraint | Equivalent Penalty |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Table 1: Few known constraint/penalty pairs (Glover, Kochenberger and Du, 2019).

Consider a problem where x1 or x2 should be equal to 1, but they cannot be identical.

Using the equivalent penalty from Table 1, we can express our objective function as shown in (3.7). The penalty is larger than zero only when both decision variables are identical.

## 3.3 Natural QUBO Formulation

Some CO problems can be naturally expressed in QUBO. In this subsection, we will summarize one such problem described by Glover, Kochenberger and Du (2019) to demonstrate the ‘*natural*’ QUBO formulation.

### 3.3.1 Minimum Vertex Cover

Minimum Vertex Cover (MVC) is a CO problem, where we are given an undirected graph with vertexes and edges represented by sets *V* and *E*. A vertex ‘covers’ the edges that it is incident to. *Vertex cover* is a subset of *V* that covers all the edges. Figure 2 demonstrates when a subset is a vertex cover. The objective is to find a vertex cover with the minimum number of vertices.

Diagram, schematic

Description automatically generated

Figure 2: Visual representation of normal subsets and a vertex cover.

We can express decision vector *x* in the following way: if vertex *j* is in the vertex cover xj = 1, otherwise xj = 0. Then minimisation function, before we consider constraints, is the sum of all vertices in the cover (3.8).

To get a feasible solution, we need to cover all the edges. In other words, every edge should have at least one vertex that belongs to the vertex cover.

Table 1 contains a penalty equivalent to this constraint. Using it, we can make the following minimisation function:

This formula can be expressed as , where *c* – is a constant that has no influence on the optimisation and can be removed. This leaves us with an unconstrained QUBO model.

### 3.3.2 Example Solution

MVC to be solved is shown in Figure 3.

Schematic

Description automatically generated with low confidence

Figure 3: Example MVC graph.

Using (3.10), we can express and then transform this problem in the following way:

If we remove the constant *6M*, which does not affect optimisation, and represent (3.12) using matrixes, we will get the following QUBO:

By assigning an arbitrary value of 8 to the *M*, we get the following solution:

A close-up of a stethoscope

Description automatically generated with medium confidence

Figure 4: A solution to the example MVC.

## 3.4 Non-Natural QUBO Formulation and Slack Variables

Not all CO problems have a natural QUBO formulation, and equivalent penalties for some constraints are unknown. Such problems can be formulated in QUBO too. In the first part of this subsection, we will show a general approach to QUBO formulation. The second part will focus on how inequality constraints can be expressed as quadratic penalties by introducing slack variables. Both the general approach and the slack variables have been accurately described by Glover, Kochenberger and Du (2019).

### 3.4.1 QUBO Formulation

Consider a constrained minimisation problem. Without quadratic penalties, it will have the following form:

Where *x* is a binary decision vector and *C* is a square matrix. If the constraints consist of integers, they can be expressed as , which is explained in (3.4.2). Then, the quadratic penalty added to the original objective function is . The resulting function with penalties is shown in (3.16).

Constant *c*, which does not affect the optimisation, is removed, and we get a QUBO model.

### 3.4.2 Slack Variables

Consider the following inequality constraint:

It can be transferred into an equality constraint by adding a slack variable, *s*, as shown in (3.18)*.*

By using binary expansion on *s*, we will get the equality shown in (3.19). It satisfies all configurations of decision variable *x*.

As both *xj* and *sj* are binary, they can be contained in vector *x*, and coefficients of x can be held in a single matrix *A*. The inequality is then in a standard QUBO form: .

## 3.5 Algorithms for Solving QUBOs

Many algorithms can effectively solve problems in QUBO formulation (Kochenberger and Glover, 2006). We have already described quantum annealing and the digital annealer algorithm in section 2. This subsection will collect some of the algorithms available in programming libraries dedicated to solving QUBO models.

### 3.5.1 Simulated Annealing

We already described SA in section 1. When solving QUBOs, neighbouring nodes are located *n* bit-flips away from the current node, where *n* is usually 1 (Alkhamis, Hasan and Ahmed, 1998). It is implemented in the ‘dwave-neal’ (2015) library.

### 3.5.2 Tabu Search

Tabu search has been adapted to be used with QUBO models (Palubeckis, 2004). It can quickly find minima in neighbourhoods but may sometimes struggle to escape them. It is implemented in the ‘dwave-tabu’ (2018) library.

### 3.5.3 Decomposing Solver

Decomposing solver splits large QUBO into smaller subQUBOs, solves them separately and combines the results (Booth, Reinhardt and Roy, 2017). The subQUBOs can be solved using a D-Wave QA device or tabu search with a classical machine. It is implemented in the ‘qbsolv’ (2017) library.

This approach is practical when using quantum devices because larger QUBOs cannot be mapped on them, but smaller subQUBOs can be, allowing us to solve bigger problems. On classical devices, we achieve a speedup by decomposing large QUBOs before applying tabu search.

### 3.5.4 Steepest Descent Algorithm

This is a greedy algorithm similar to gradient descent, but instead of making moves based on the gradient, it uses local minimisation. At each step, we move to the state one bit-flip away that causes the highest energy drop. It is implemented in the ‘dwave-greedy’ (2019) library.

# 4 Penalty Coefficient Optimisation Techniques

In this section, we will present four techniques of penalty coefficient optimisation, two of which are state-of-art. This area of research is relatively new, and the number of unique approaches to this problem is limited.

Penalty coefficient optimisation is not trivial (Verma and Lewis, 2020). If the chosen coefficient is too low, the found solution may be infeasible. If it is too large, penalties will dominate the search space, which erases the difference between good and bad nodes, making the optimisation problem numerically unstable, decreasing the accuracy of the produced solutions (Vyskočil, Pakin and Djidjev, 2019).

## 4.1 Analytical

The first approach involves using domain knowledge to set suitable *M* values (Glover, Kochenberger and Du, 2019). We need to make a rough estimate of the original objective function and set the initial penalty coefficient to about 75% to 150% of this estimate. The resultant problem needs to be solved using a QUBO solver. The solution should be checked for feasibility: if feasible, we can reduce the *M* and try again; if not, we need to increase *M*. We keep solving, checking for feasibility, and updating *M* until we settle with a satisfactory penalty coefficient.

This approach is simple and can produce good penalty coefficients. However, estimating the original objective function “by eye” is not always easy, especially when the problem is large. Also, this trial-and-error procedure requires at least multiple calls to the solver, which is expensive and impractical for performance-critical applications (Huang *et al.*, 2021). This method of penalty estimation is not automated. It would be more efficient to use an algorithm to estimate the initial *M*, for example, the one by Verma and Lewis (2020), and then adjust it in a loop using the solver.

## 4.2 Numerical 1

This method involves choosing an arbitrary value of matrix *Q* and making the penalty coefficient *M* larger than it (Verma and Lewis, 2020). Although this approach is fast and straightforward, it does not always work. We can try to improve our estimated *M* using the methodology described in (Glover, Kochenberger and Du, 2019) and summarised in section (4.1), but that would bring the downsides of the analytical approach (inefficiency being one of them) to this method too.

## 4.3 Numerical 2

Verma and Lewis (2020) use the general formulation of QUBO described in section 3.4 to estimate a lower bound of *M*. They consider a problem where the next node (*xto*) has lower energy than the current node (*xfrom*). However, *xfrom* is feasible, and *xto* is not. This is shown in (4.1).

In this scenario, penalty coefficient *M* should be large enough to impose a sufficient penalty that will prevent transition, as shown in (4.2). Otherwise, we would choose an infeasible solution over a feasible one.

Equation 4.2 can be rearranged to get a lower bound of *M*:

To calculate the exact lower bound, we need to find a transition that will give us the maximum energy change with the minimum penalty imposed. Since there are no efficient methods for minimising the denominator, they make an optimistic assumption that it is always equal to 1. They then propose an algorithm that finds the maximum transition for a 1-flip solver in *O(2n2)* steps, which is the estimated lower bound of a penalty coefficient as shown in (4.4).

Both and flips can cause energy drop. For example, flipping from 0 to 1 will bring a positive transition to and flipping 1 to 0 will bring a positive transition to . Thus, we need to flip every *xi* twice to find the maximum transition:

The maximum transition of flips can be found by doing the following. For every *xi*, sum the coefficient of its linear part (*qii*) and all the positive coefficients of its quadratic part (*qij > 0*). Negative coefficients are ignored as the second variable in quadratic (*xj*) will be 0. The maximum transition of flips is found similarly, but since it brings positive changes by nullifying the negative coefficients, we need to sum coefficient negatives (will bring a positive change of 6, which is the negative of the coefficient *q1*). For the same reasons, we need to sum the negative coefficients of the quadratic part (*qij < 0*) instead of positive ones. The maximum transition encountered in this process is the estimate of *M*. All these steps are expressed in (4.6). The estimates with feasible *xto* are discarded as we only care about feasible to infeasible moves. The next largest *Mest* is selected instead*.*

This method is fast and produces good estimates of the lower bound of *M*, but it does not consider the quadratic penalties, , when estimating the penalty coefficient. That is why this approach can produce *M* values that are too large. One promising approach is to use Walsh analysis (Brownlee, 2009), which explicitly attaches energies to variable interactions in a way natural to the QUBO model. These energies may be used to estimate what values the quadratic penalties can take to decrease the lower bound estimate further.

## 4.4 Machine Learning

Huang *et al.* (2021) proposed using machine learning (ML) to optimise penalty coefficients. They have reviewed the analytical approach from section (3.2), where different penalty coefficients are tested on the QUBO solver to find the optimal value. Calling a QUBO solver multiple times to solve a single problem is expensive and is not suitable for applications where performance is critical. As an alternative, they propose using solutions obtained from the QUBO solver in the past to train an ML model called *Solver Surrogate* (SS). Then instead of calling a QUBO solver, we will call a faster SS, adjusting the coefficient repeatedly until a satisfactory result is achieved.

As conventional QUBO solvers usually return a batch of solutions with their energies, we can count how many are feasible and calculate the probability of feasibility (*Pf* ) with the penalty coefficient used. We can also calculate the mean energy of the solutions (*Eavg*) and the standard deviation (*Estd*). The SS will be trained to predict these three values.

During the training, features of the CO problem (*g*) and the penalty coefficient (denoted *A* in this paper) from the dataset will be inputted into SS to make a prediction, which is used with ground truth to calculate the loss. The loss then updates the model through backpropagation. This is repeated with the entire dataset multiple times to train the SS. Figure 5 demonstrates the described process.

Diagram

Description automatically generated

Figure 5: Solver Surrogate training (Huang et al., 2021).

After training, the surrogate is used to find penalty coefficient estimates. Problem features and an initial *A* are inputted into SS to predict *Pf*, *Eavg* and *Estd*. Using the predicted values, *we* can adjust *A* and rerun the SS. Huang *et al.* (2021) propose three strategies (*Parameter Selection Strategies*) for adjusting *A*. This process is repeated until a suitable estimate of the penalty coefficient (*Ã*) is found. The described steps are demonstrated in Figure 6.

Diagram

Description automatically generated

Figure 6: Estimating penalty coefficient with Solver Surrogate (Huang et al., 2021).

This method benefits from good penalty coefficient estimates found quicker than if we made calls to QUBO solver. It can be used to solve many variations of the same CO problem, which is common in industry (Fujitsu, no date; D-Wave Systems, 2021). However, it will be impossible to train the model if we do not have a dataset with the problems solved in the past, their solutions and penalty coefficients used. In this case, we will need another strategy to estimate penalty coefficients while data collection takes place. The method from (4.3) could be used for this purpose.

# 5 Summary

# References

Alkhamis, T. M., Hasan, M. and Ahmed, M. A. (1998) ‘Simulated annealing for the unconstrained quadratic pseudo-Boolean function’, *European Journal of Operational Research*, 108(3), pp. 641–652. doi: 10.1016/S0377-2217(97)00130-6.

Aramon, M. *et al.* (2019) ‘Physics-Inspired Optimisation for Quadratic Unconstrained Problems Using a Digital Annealer’, *Frontiers in Physics*, 0(APR), p. 48. doi: 10.3389/FPHY.2019.00048.

Bertsimas, D. and Tsitsiklis, J. (1993) ‘Simulated Annealing’, *Statistical Science*, 8(1), pp. 10–15.

Booth, M., Reinhardt, S. P. and Roy, A. (2017) ‘Partitioning Optimization Problems for Hybrid Classical / Quantum Execution TECHNICAL REPORT’, *D-Wave Technical Report Series*, 14-1006A-A.

Boyd, J. (2018) ‘Fujitsu’s CMOS Digital Annealer Produces Quantum Computer Speeds’, *IEEE Spectrum*, 28 May. Available at: https://spectrum.ieee.org/ieee-courses-on-digital-transformation (Accessed: 12 November 2021).

Brownlee, A. E. I. (2009) ‘Multivariate Markov networks for fitness modelling in an estimation of distribution algorithm.’ Available at: https://rgu-repository.worktribe.com/output/247858/multivariate-markov-networks-for-fitness-modelling-in-an-estimation-of-distribution-algorithm (Accessed: 18 November 2021).

Collins, E. F. (1921) ‘ELECTRICALLY HEATED GLASS ANNEALING LEHR1’, *Journal of the American Ceramic Society*, 4(5), pp. 335–349. doi: 10.1111/J.1151-2916.1921.TB18664.X.

Crosson, E. and Harrow, A. W. (2016) ‘Simulated Quantum Annealing Can Be Exponentially Faster Than Classical Simulated Annealing’, *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, 2016-Decem, pp. 714–723. doi: 10.1109/FOCS.2016.81.

Cruz-Santos, W., Venegas-Andraca, S. E. and Lanzagorta, M. (2019) ‘A QUBO Formulation of Minimum Multicut Problem Instances in Trees for D-Wave Quantum Annealers’, *Scientific Reports 2019 9:1*, 9(1), pp. 1–12. doi: 10.1038/s41598-019-53585-5.

D-Wave Systems (2021) *Volkswagen: Navigating Tough Automotive Tasks with Quantum Computing*. Available at: www.dwavesys.com/d-wave-launch (Accessed: 5 October 2021).

Dattani, N., Szalay, S. and Chancellor, N. (2019) ‘Pegasus: The second connectivity graph for large-scale quantum annealing hardware’.

‘dwave-greedy’ (2019). Available at: https://github.com/dwavesystems/dwave-greedy (Accessed: 17 November 2021).

‘dwave-neal’ (2015). Available at: https://github.com/dwavesystems/dwave-neal (Accessed: 17 November 2021).

‘dwave-tabu’ (2018). Available at: https://github.com/dwavesystems/dwave-tabu (Accessed: 17 November 2021).

Farhi, E. *et al.* (2001) ‘A Quantum Adiabatic Evolution Algorithm Applied to Random Instances of an NP-Complete Problem’, *Science*, 292(5516), pp. 472–476. doi: 10.1126/SCIENCE.1057726.

Fujitsu (no date) *Reducing traveling distance for parts picking operations by up to 45%*. Available at: https://www.fujitsu.com/global/services/business-services/digital-annealer/case-studies/201804-fjit.html (Accessed: 5 October 2021).

Glover, F., Kochenberger, G. and Du, Y. (2019) ‘Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models’, *4OR*, 17(4), pp. 335–371. doi: 10.1007/S10288-019-00424-Y.

Higham, C. F. and Bedford, A. (2021) ‘Quantum Deep Learning: Sampling Neural Nets with a Quantum Annealer’. Available at: https://arxiv.org/abs/2107.08710v1 (Accessed: 11 November 2021).

Huang, T. *et al.* (2021) ‘QROSS: QUBO Relaxation Parameter optimisation via Learning Solver Surrogates’, *2021 IEEE 41st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 35–40. doi: 10.1109/ICDCSW53096.2021.00013.

Johnson, D. S. *et al.* (1989) ‘Optimisation by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning’, 37(6).

Johnson, M. W. *et al.* (2011) ‘Quantum annealing with manufactured spins’, *Nature 2011 473:7346*, 473(7346), pp. 194–198. doi: 10.1038/nature10012.

Katzgraber, H. G. and Novotny, M. A. (2018) ‘How Small-World Interactions Can Lead to Improved Quantum Annealer Designs’, *Physical Review Applied*, 10(5), p. 054004. doi: 10.1103/PHYSREVAPPLIED.10.054004/FIGURES/13/MEDIUM.

Kirkpatrick, S., Gelatt, ; C D and Vecchi, ; M P (1983) ‘Optimisation by Simulated Annealing’, *New Series*, 220(4598), pp. 671–680.

Kochenberger, G. *et al.* (2014) ‘The unconstrained binary quadratic programming problem: A survey’, *Journal of Combinatorial Optimization*, 28(1), pp. 58–81. doi: 10.1007/S10878-014-9734-0/TABLES/1.

Kochenberger, G. A. and Glover, F. (2006) ‘A Unified Framework for Modeling and Solving Combinatorial Optimisation Problems: A Tutorial’, *Multiscale Optimization Methods and Applications*, pp. 101–124. doi: 10.1007/0-387-29550-X\_4.

Liang, F., Cheng, Y. and Lin, G. (2014) ‘Simulated Stochastic Approximation Annealing for Global Optimization With a Square-Root Cooling Schedule’, *https://doi.org/10.1080/01621459.2013.872993*, 109(506), pp. 847–863. doi: 10.1080/01621459.2013.872993.

Lucas, A. (2014) ‘Ising formulations of many NP problems’, *Frontiers in Physics*, 0, p. 5. doi: 10.3389/FPHY.2014.00005.

Ohzeki, M. *et al.* (2019) ‘Control of Automated Guided Vehicles Without Collision by Quantum Annealer and Digital Devices’, *Frontiers in Computer Science*, 1, p. 9. doi: 10.3389/FCOMP.2019.00009/BIBTEX.

Ohzeki, M. and Nishimori, H. ‘Quantum annealing: An introduction and new developments’, *Journal of Computational and Theoretical Nanoscience*, 8(6), pp. 963–971. doi: 10.1166/JCTN.2011.1776963.

Palubeckis, G. (2004) ‘Multistart Tabu Search Strategies for the Unconstrained Binary Quadratic Optimization Problem’, *Annals of Operations Research 2004 131:1*, 131(1), pp. 259–282. doi: 10.1023/B:ANOR.0000039522.58036.68.

Poole, D. L. and Mackworth, A. K. (2017) ‘Artificial Intelligence: Foundations of Computational Agents’, pp. 136–137. doi: 10.1017/9781108164085.

‘qbsolv’ (2017). Available at: https://github.com/dwavesystems/qbsolv (Accessed: 17 November 2021).

Şeker, O. *et al.* (2020) ‘Digital Annealer for quadratic unconstrained binary optimisation: a comparative performance analysis’. Available at: http://arxiv.org/abs/2012.12264 (Accessed: 3 October 2021).

Snelling, D. *et al.* (2020) ‘A Quantum-Inspired Approach to De-Novo Drug Design’. doi: 10.26434/CHEMRXIV.12229232.V1.

Tanaka, S. and Tamura, R. (2012) ‘QUANTUM ANNEALING AND QUANTUM FLUCTUATION EFFECT IN FRUSTRATED ISING SYSTEMS’.

Verma, A. and Lewis, M. (2020) ‘Penalty and partitioning techniques to improve performance of QUBO solvers’, *Discrete Optimization*, p. 100594. doi: 10.1016/j.disopt.2020.100594.

Vyskočil, T., Pakin, S. and Djidjev, H. N. (2019) ‘Embedding Inequality Constraints for Quantum Annealing Optimization’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11413 LNCS, pp. 11–22. doi: 10.1007/978-3-030-14082-3\_2.

Wu, H. and Fan, G. (2020) ‘An overview of tailoring strain delocalisation for strength-ductility synergy’, *Progress in Materials Science*, 113, p. 100675. doi: 10.1016/J.PMATSCI.2020.100675.