Literature *Review*

Raufs Dunamalijevs

Table of Contents

1 Annealing in Combinatorial Optimization

In this section we will discuss a class of metaheuristics that are inspired by the physical process of *annealing*. At the beginning, we will explain what it is. Then we will talk about multiple annealing metaheuristics that are relevant to this work. Finally, we will look at hardware named Digital Annealer.

1.1 Annealing

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

1.2 Simulated Annealing

Simulated annealing (SA) is a metaheuristic that uses the concept of temperature to approximate global optimum of a combinatorial optimization 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 the optimization with a high temperature and decrease it at a controlled rate. If that rate is low enough, we are guaranteed to find a global optimum (Liang, Cheng and Lin, 2014). However, it is possible to find good approximations of the optima significantly faster at higher cooling rates.

The temperature concept allows the algorithm to explore the search space and find the optimal solution (Poole and Mackworth, 2017). The SA starts with a high temperature. At that point the algorithm acts like random walk, choosing random moves with high probability, which prevents us from being stuck at local optima – this is when exploration happens. At the end of the SA, when the temperature is low, the algorithm prioritizes advantageous moves. It acts like greedy algorithm and tries to find the best solution. As we shift from higher to lower temperatures, we gradually shift from exploring the search space to find the optimum.

Let c be current node, n be next node (neighbour of node c) and $evaluation(x)$ be the function that evaluates the energy of the node x. Then the energy transition (ΔE) can be defined by the following formula:

$$\Delta E = evaluation(n) - evaluation(c)$$

ΔE and T are the variables that will affect the probability of choosing node n, where ΔE is an energy transition (negative when minimising) that the proposed node will introduce, and T is controlling how ΔE influences the probability of choosing this node (Johnson *et al.*, 1989).

$$x = \frac{\Delta E}{T}$$

The *acceptance probability function* is the function that maps current node, proposed node, and Temperature to a probability. There are many functions that suit this purpose, the original being an exponential function (Kirkpatrick, Gelatt and Vecchi, 1983).

$$P(c, n, T) = e^{-\frac{\Delta E}{T}}$$

When maximizing, the negative sign should be removed from the formula.
If that acceptance probability is higher than a uniformly generated random number, then node c is rejected in favour of node n. However, when $\Delta E \leq 0$ (downhill movement), we will always choose the proposed node (Kirkpatrick, Gelatt and Vecchi, 1983).

$$\Delta E \leq 0 \begin{cases} False & P(c,n,T) > random(0,1) \begin{cases} False & continue \\ True & c \leftarrow n \end{cases} \\ True & c \leftarrow n \end{cases}$$

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
```

1.3 Quantum Annealing

Quantum annealing (QA) is a metaheuristic that uses quantum-mechanical fluctuations to find a global minimum of a function (Ohzeki and Nishimori, 2010; Tanaka and Tamura, 2012). In QA we start with a strong quantum field and gradually decrease its strength (quantum fluctuation) to find a ground state that is optimal. This is analogous to SA, where we decreased the temperature (thermal fluctuation) to achieve the same result.

QA begins in a uniform superposition of all possible states, but as we decrease the strength of quantum field, we approach a single solution state (Farhi *et al.*, 2001).
In SA neighbouring nodes are located 1 move away for the current node. In QA the neighbouring nodes are in a certain range. This range is defined by the strength of the quantum field, slowly ceasing in the process of annealing. Transition from one state to another can happen within that range. If there is a short wall between current state, which happens to be a local minimum, and

the global minimum, 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 compared to SA when dealing with search spaces with many thin barriers (Crosson and Harrow, 2016). This effect is called *quantum tunnelling*. Comparison of thermal hopping and quantum tunnelling can be seen on Figure 1.

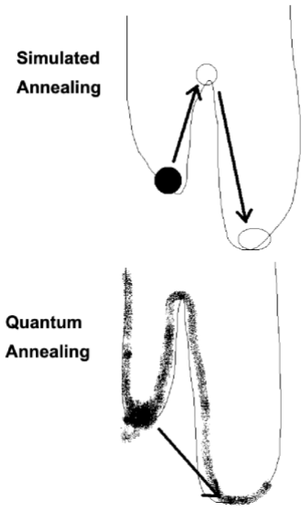

Figure 1. Comparison of node transitions in Simulated and Quantum Annealing.

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

1.4 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 3 major differences:

1. Instead of considering a single neighbouring node and choosing it if the acceptance probability is higher than a uniform random number, DA considers all neighbouring nodes in parallel. If more than a single node was accepted, 1 of them will be chosen uniformly at random to move into. This is advantageous as in SA if the node was rejected, we need to generate and then consider a new one from the same position, which takes time. In DA, however, all the nodes are considered in parallel and the acceptance probability per "turn" is therefore higher.
2. DA uses *dynamic offset*, which means that if no nodes were accepted, the next acceptance probability will be artificially increased to help the algorithm overcome narrow barriers.
3. All runs begin with the same random state to save time (in SA that runs in parallel the initial states are generated individually for each of the runs). Otherwise, it would need to evaluate the new initial node and all the new neighbouring nodes for each run.

These differences can be observed in DA pseudocode (Aramon *et al.*, 2019):

```
initial state ← random state
for each run
	current state ← initial state
	E_offset ← 0
	for each step
		update temperature if needed
		for each neighbour (in parallel)
			calculate ΔE_neighbour
			consider neighbour using ΔE_neighbour - ΔE_offset
			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)
			E_offset ← 0
		else
			increase E_offset
		end
	end
end
```

Although it is hard to make comparisons as the technology is quickly developing, it has been shown that DA can be faster and produce more precise solutions compared to D-Wave 2000Q (penultimate generation of D-Wave's QA devices) in certain scenarios (Ohzeki *et al.*, 2019). DA's are used in industry and science (Fujitsu, no date; Snelling *et al.*, 2020)

Currently QA devices are expensive and difficult to run as the technology is still under development (Şeker *et al.*, 2020). D-Wave 2000Q also suffers from qubit noise that results in lower precision (Katzgraber and Novotny, 2018), and sparse connectivity between spins that makes it less effective (Hamerly *et al.*, 2018). These problems are actively worked on and some of them are addressed in more recent QA devices (Dattani, Szalay and Chancellor, 2019). As QA technology develops, it will likely outperform DA largely due to its quantum parallelism (Boyd, 2018).

2 QUBO

Quadratic Binary Unconstrained Optimization (QUBO) is a mathematical formulation that can be applied to many combinatorial optimization (CO) problems (Kochenberger *et al.*, 2014). Problems formulated in QUBO can be subsequently solved using QUBO solvers. QA devices developed by D-Wave can solve CO problems that are formulated as QUBO or Ising models (Cruz-Santos, Venegas-Andraca and Lanzagorta, 2019). It has been shown that QUBO and Ising models are equivalent (Lucas, 2014), therefore one can be derived from another. DA's can only solve problems that are formulated in QUBO (Aramon *et al.*, 2019).
There are many combinatorial optimization problems from industry, government and science that can be reformulated in QUBO (Glover, Kochenberger and Du, 2019).

In this section we will discuss the structure of QUBO, basics of reformulating constrained CO problems into equivalent unconstrained QUBO models, algorithms that can be used to solve these problems and relevant programming libraries that can be useful when dealing with QUBO's. We will introduce the concept of penalty, which will be explained in greater details in the next section.

2.1 QUBO model

QUBO models have been described in great details in the work of Glover, Kochenberger and Du (2019). This subsection will summarize the key information relevant to this project.

The QUBO model can be expressed in the following way:

$$minimize/maximize\ y = x^t Q x$$
$$\forall z \in x, z \in \{0,1\}$$

where y is the value to be optimized for, x is a vector of decision variables and Q is a square matrix with coefficients. QUBO models are unconstrainted, the only restriction being that every variable in decision vector x should be either 0 or 1. It is self-contained as all the information needed for the optimisation is stored in the matrix Q. QUBO problems are NP-hard.

A simple example of how to convert a Boolean function into QUBO can be demonstrated with the following minimization problem:

$$y = -5x_1 - 3x_2 - 8x_3 - 6x_4 + 4x_1x_2 + 8x_1x_3 + 2x_2x_3 + 10x_3x_4$$

Our function has 2 parts: linear and quadratic. As every x_j belongs to {0, 1}, we can easily make the linear part quadratic (because $0^2=0$ and $1^2=1$), making the entire function quadratic.

$$y = -5x_1^2 - 3x_2^2 - 8x_3^2 - 6x_4^2 + 4x_1x_2 + 8x_1x_3 + 2x_2x_3 + 10x_3x_4$$

We can then write it using matrixes in the form of $y = x^t Q x$, which is a QUBO model formula shown at the beginning.

$$y = x_1 x_2 x_3 x_4 \begin{bmatrix} -5 & 2 & 4 & 0 \\ 2 & -3 & 1 & - \\ 4 & 1 & -8 & 5 \\ 0 & 0 & 5 & -6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

2.2 Constraints and Penalties

In the previous subsection it has been shown how an unconstrained minimization problem can be reformulated as a QUBO. Constrained problems can be reformulated in this model too. The constraints will be expressed as quadratic penalties which will be added to the original objective function (Glover, Kochenberger and Du, 2019). The more constraints are broken, the larger is the overall penalty imposed. When none of the constraints are broken, the penalty imposed will be equal to 0.
Since we are trying to solve a minimization problem, solutions that brake constraints and impose penalty on the objective function will be avoided.

$$y = original\ objective\ function + M * quadratic\ penalties$$

M is a positive penalty scalar (Verma and Lewis, 2020). It controls the effect that the broken constraints will have on the objective function. If we have a soft constraint that is allowed to be broken, the penalty scalar can be decreased, which will decrease the effect that the broken constraint will have on our function. It is possible to use multiple scalars for various constraints if they are of a different importance. But usually only a single M is used (Glover, Kochenberger and Du, 2019).

If the penalty coefficient is too low, the broken constraints will be undervalued, and the solution produced by the optimizer will be infeasible. On the other hand, if the penalty coefficient is too large, the solution process will be negatively impacted as 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 in some cases (Glover, Kochenberger and Du, 2019), but it has also been shown that some problems can have solutions of better quality if you choose penalty coefficient carefully (Şeker, Tanoumand and Bodur, 2020). Finding an optimal penalty coefficient value is not trivial and different techniques have been proposed to estimate it (Verma and Lewis, 2020; Huang et al., 2021).

Quadratic penalties for certain constraint types are already known. Some of them are shown on the Table 1.

Classical Constraint	Equivalent Penalty
$x + y \leq 1$	$P(xy)$
$x + y \geq 1$	$P(1 - x - y + xy)$
$x + y = 1$	$P(1 - x - y + 2xy)$
$x \leq y$	$P(x - xy)$
$x_1 + x_2 + x_3 \leq 1$	$P(x_1x_2 + x_1x_3 + x_2x_3)$
$x = y$	$P(x + y - 2xy)$

Table of a few Known constraint/penalty pairs

Table 1. Some classical constraints and equivalent quadratic penalties.

Consider a problem where either x_1 or x_2 should be equal to 1, but they cannot be equal to 0 or 1 together. This constraint can be described with the following formula:

$$x_1 + x_2 = 1$$

Using the penalty equivalent to our constraint, we can express our objective function in the following way:

$$y = f(x) + M(1 - x_1 - x_2 + 2x_1x_2)$$

where f(x) is the original objective function. We can see that the penalty is imposed only when both decision variables are equal to 0 or 1, that is when the constraint is broken.

2.3 Natural QUBO Formulation

There exist combinatorial optimization problems that can be naturally expressed as QUBO models. Some of them have been described by Glover, Kochenberger and Du (2019). One of such problems will be summarized in this subsection using their work to demonstrate the process of the '*natural*' QUBO formulation.

2.3.1 Minimum Vertex Cover

Minimum Vertex Cover (MVC) is a CO problem, where we are given undirected graph with vertexes and edges represented by sets *V* and *E* respectively. A single vertex 'covers' the edges that it is incident to. *Vertex cover* is a subset of vertexes *V*, which covers all the edges *E*. Figure 2 compares vertex cover and standard subsets of *V*. In MVC we aim to find a vertex cover with the minimum number of vertices.

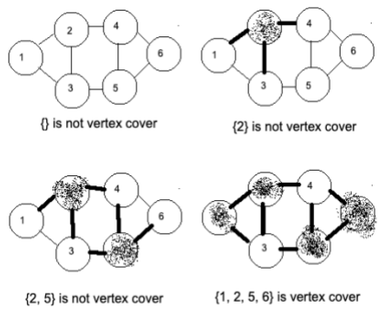

Figure 2. Visual representations of common subsets of V and a vertex cover.

The decision vector x can be expressed in the following way. If vertex j is in the vertex cover, then is $x_j = 1$. Otherwise, $x_j = 0$. Then our minimization function, before we consider any constraints, can be defined as the sum of all vertices that are in the cover.

$$\forall x_j \in x, x_j \in \{0,1\}$$
$$f(x) = \sum_{j \in V} x_j$$

To get a feasible solution, we need to cover all the edges. This can be expressed as the following constraint: every edge should have at least 1 vertex that belongs to the vertex cover. Because if it does not, the edge under consideration is not covered and, therefore, not all the edges are covered. This is infeasible. Thus, the constrains are:

$$(\forall i, \forall j) \in E, x_i + x_j \geq 1$$

Table 1 contains a penalty equivalent to this type of constraints. Using it we can make the following minimization function:

$$y = \sum_{j \in V} x_j + M \sum_{(i,j) \in E} 1 - x_i - x_j + x_i x_j$$

This formula can be expressed as $y = x^t Q x + c$, where c – is a constant that has no influence on the optimisation process and can be removed. This leaves us with an unconstrained QUBO model.

2.3.2 Example Solution

We are given the following graph which we need to find an MVC for:

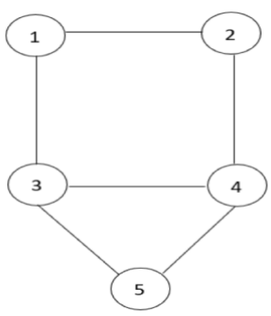

Figure 3. Example MVC graph.

Using the formula derived previously, we can express and then transform the problem in the following way:

$$\begin{aligned} y = {} & x_1 + x_2 + x_3 + x_4 + x_5 + \\ & M(1 - x_1 - x_2 + x_1x_2) + \\ & M(1 - x_1 - x_3 + x_1x_3) + \\ & M(1 - x_2 - x_4 + x_2x_4) + \\ & M(1 - x_3 - x_4 + x_3x_4) + \\ & M(1 - x_3 - x_5 + x_3x_5) + \\ & M(1 - x_4 - x_5 + x_4x_5) + \end{aligned}$$

$$\begin{aligned} y = {} & (1 - 2M)x_1 + (1 - 2M)x_2 + (1 - 3M)x_3 + (1 - 3M)x_4 + (1 - 2M)x_5 + Mx_1x_2 + Mx_1x_3 \\ & + Mx_2x_4 + Mx_3x_4 + Mx_3x_5 + Mx_4x_5 + 6M \end{aligned}$$

If we remove the constant $6M$, which has no effect on the optimization (we can add it after the optimization has finished if needed), from the end of the formula, and assign an arbitrary value of 8 to the penalty scalar M, we will get the following QUBO:

$$Q = \begin{bmatrix} -15 & 4 & 4 & 0 & 0 \\ 4 & -15 & 0 & 4 & 0 \\ 4 & 0 & -23 & 4 & 4 \\ 0 & 4 & 4 & -23 & 4 \\ 0 & 0 & 4 & 4 & -15 \end{bmatrix}$$

$$y = x^t Q x$$

With the following solution:

$$V = \{2, 3, 5\}, y = -45, x = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

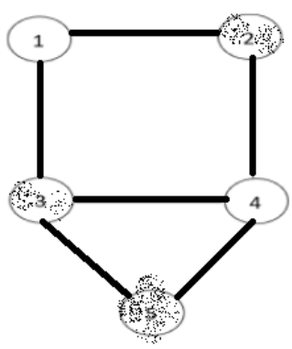

Figure 4. Solution to the example MVC.

2.4 Non-Natural QUBO Formulation and Slack Variables

Not all CO problems have a natural QUBO formulation. Also, quadratic penalties for some constrains might be 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. In the second part, we will show how any inequality constraint can be expressed as a quadratic penalty by introducing slack variables. Both processes have been accurately described by Glover, Kochenberger and Du (2019).

2.4.1 QUBO Formulation

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

$$y = x^t C x$$
$$\forall z \in x, z \in \{0,1\}$$

where x is a binary decision vector and C is a square matrix. If the constraints of the problem consist of integers, they can always be written as (this will be explained in more details in the subsection denoted to slack variables):

$$Ax = b$$

Then the quadratic penalties that we need to add to the original objective function are:

$$quadratic\ penalties(x) = M(Ax - b)^t(Ax - b)$$

And the resultant objective function with constraints will be:

$$y = x^t C x + M(Ax - b)^t(Ax - b)$$
$$y = x^t C x + x^t D x + c$$
$$y = x^t Q x + c$$

Constant c can be removed as it has no effect on the optimization (it can be added after a solution is found if needed). That is how we get a standard $y = x^t Q x$ QUBO model out of a general constrained problem that has no natural QUBO formulation.

2.4.2 Slack Variables

Consider the following inequality constraint:

$$4x_1 + 5x_2 - x_3 \leq 6$$

It can be transferred into an equality constraint by adding a slack variable, s. The slack variable cannot be bigger than 7 because the lowest number that the left side of the inequality takes is -1 with (0,0,1) configuration. It also cannot be smaller than -3 as the largest number that the left side takes is 9 with (1,1,0) configuration.

$$4x_1 + 5x_2 - x_3 + s = 6$$
$$-3 \leq s \leq 7$$

If we express s using binary expansion, which will satisfy all configurations of decision variable x, we will get the following equality:

$$4x_1 + 5x_2 - x_3 + s_1 + 2s_2 + 4s_3 = 6$$
$$\forall z \in s, z \in \{0,1\}$$

As both x_j and s_j are binary at this point, they can both be contained in a single vector: x. And all the coefficients can then be contained in a single matrix A.

$$Ax = b$$

Then, as shown in the previous part of this subsection, this constraint can be expressed as a quadratic penalty $M(Ax - b)^t(Ax - b)$ and subsequently used to achieve a standard QUBO model.

2.5 Algorithms for solving QUBOs

> the techniques mentioned in Section 2.1. QUBO problems are hard both in theory and practice, so that even very efficient exact algorithms can handle up to 500 variables (Mauri and Lorena 2012).

https://arxiv.org/pdf/2012.12264.pdf

Survey includes algorithms
https://sci-hub.3800808.com/10.1007/s10878-014-9734-0

3 Penalty Coefficient Optimisation Techniques

In this section we will present multiple techniques of penalty coefficient optimisation. Two of them will be standard and the other two will be state-of-art. This area of research is relatively new and the number of unique approaches to this problem is limited.

As mentioned in previous sections, penalty coefficient optimisation is not trivial (Verma and Lewis, 2020). If the coefficient chosen is too low, the solution to the problem will be infeasible. If it is too large, penalties will dominate the search space, erasing the difference between good and bad states. This makes the optimisation problem numerically unstable decreasing the accuracy of produced solutions (Vyskočil, Pakin and Djidjev, 2019).

3.1 Analytical

The first approach is analytical and involves using our domain knowledge to set a suitable M value (Glover, Kochenberger and Du, 2019). At the beginning, we need to make a rough estimate of the original objective function. Then, we set our penalty coefficient M to about 75% to 150% of this estimate. The resultant problem needs to be solved using QUBO solver, and the solution should be checked for feasibility. If feasible, we can try to reduce the M and try again (or keep this coefficient). If it is not, we need to increase M. We then repeat the process of producing solution with the chosen M, checking for feasibility, and updating M until we settle with a penalty coefficient that satisfies us.

The benefit of this approach is its simplicity and that by the end of it we get a satisfiable penalty coefficient. But at the same time, it is not always easy to estimate the original objective function “by eye”, especially when the problem is large.
Also, this trial-and-error procedure requires at least multiple calls to the solver. This is expensive and impractical for performance critical applications (Huang *et al.*, 2021).
This method of penalty estimation is not automated. If we are trying to solve many problems, it is not efficient to estimate penalty coefficient manually.
And even though this approach is somewhat systematic, it is not reproducible. Thus, we can get different penalty coefficients if we apply this method on the same problem multiple times.

3.2 Numerical 1

This method involves choosing an arbitrary value of matrix Q and making the penalty coefficient M larger than that number (Verma and Lewis, 2020).

Although this approach is fast and simple, it does not always work. We can try to improve our estimated M using the methodology used in the analytical approach (Glover, Kochenberger and

Du, 2019) described in previous subsection, but that would impose the downsides of the analytical approach (inefficiency being one of them) on this method too.

3.3 Numerical 2

This method was proposed by Verma and Lewis (2020). They use general QUBO model with penalties, $\min x^t Qx + M(Ax - b)^2$, to estimate a lower bound of M.

First, they consider a problem, where the next node, x_{to}, is better than the current node, x_{from}, but the current node is feasible and the next one is not:

$$x_{from}^t Q x_{from} > x_{to}^t Q x_{to}, Ax_{from} = b \text{ and } Ax_{to} \neq b.$$

In this scenario, penalty coefficient M should be large enough to prevent such transition, because otherwise we would choose an infeasible solution over a feasible one:

$$x_{from}^t Q x_{from} < x_{to}^t Q x_{to} + M(Ax_{to} - b)^2$$

The formula can be rearranged to get a lower bound of M:

$$M > \frac{x_{from}^t Q x_{from} - x_{to}^t Q x_{to}}{(Ax_{to} - b)^2}$$

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 calculating the denominator, they assume that it is equal to 1, which is the lowest and best value it can take.

$$M_{est} = \max (x_{from}^t Q x_{from} - x_{to}^t Q x_{to})$$

They propose an algorithm that can find the maximum transition for 1-flip solver in $O(2n^2)$ steps.

Both $0 \rightarrow 1$ and $1 \rightarrow 0$ flips can result in a positive transition. For example, flipping x_1 from 0 to 1 will bring a positive transition to $y = 6x_1$ and flipping 1 to 0 will bring a positive transition to $y = -6x_1$. That is why we must consider both flips.
We need to flip every x_i of the binary decision vector x from 0 to 1 and vice-verse to find the maximum transition:

$$\max transition = max \forall x_i (\Delta(x^t Qx)|_{x_i:0\rightarrow 1}, \Delta(x^t Qx)|_{x_i:1\rightarrow 0})$$

The maximum of $0 \rightarrow 1$ (left expression in the brackets) part can be found by doing the following: for every x_i, we need to take the coefficient of its linear part (q_{ii}) and add it to the sum of all the positive coefficients that the x_i is associated with in its quadratic part ($q_{ij} > 0$). We only

consider positive coefficients as when they are negative the second decision variable that it is associated with (x_j) will take value 0, because we are looking for the maximal transition x_i can possibly bring.

The maximum of $1 \rightarrow 0$ part can be found in the same way, but because such transition brings positive changes by nullifying the negative coefficients, we need to sum their negatives ($-6x_1|_{x_1:1\rightarrow 0}$ will bring a positive change of 6, which is the negative of the coefficient q_1). For the same reasons, instead of summing the positive coefficients of the quadratic part, we need to sum the negative ones ($q_{ij} < 0$).

All the steps described can be expressed mathematically to find the estimated penalty coefficient:

$$M_{est} = \max\{q_{ii} + \sum_{q_{ij}>0}^{j\neq i} q_{ij} \forall i \in \{1,n\}, -q_{ii} - \sum_{q_{ij}<0}^{j\neq i} q_{ij} \forall i \in \{1,n\}\}$$

The estimates for x_{to} nodes that are feasible (i.e., they have no constraints associated with them) will be discarded as their penalty ($Ax_{to} - b$) will be equal to 0 and the next largest M_{est} *will be used.*

This method is fast and produces good estimates of the lower bound of M. However, it does not consider the quadratic penalties, $(Ax_{to} - b)^2$, when estimating the penalty coefficient. That is why this approach can produce M values that are too large.

3.4 Machine Learning

A is *M*
https://arxiv.org/pdf/2103.10695v1.pdf

4 Summary

Summarize everything and form a research question (or repeat the one from project proposal).

References

Aramon, M. *et al.* (2019) 'Physics-Inspired Optimization 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.

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).

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'.

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'.

Farhi, E. *et al.* (2001) 'A Quantum Adiabatic Evolution Algorithm Applied to Random Instances of an NP-Complete Problem'.

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.

Hamerly, R. *et al.* (2018) 'Scaling advantages of all-to-all connectivity in physical annealers: the Coherent Ising Machine vs. D-Wave 2000Q'.

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) 'Operations Research Society of America This content downloaded from 128', 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) 'Optimization 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.

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. (2010) 'Quantum annealing: An introduction and new developments'.

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

Şeker, O. *et al.* (2020) 'Digital Annealer for quadratic unconstrained binary optimization: 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 delocalization for strength-ductility synergy', *Progress in Materials Science*, 113, p. 100675. doi: 10.1016/J.PMATSCI.2020.100675.