

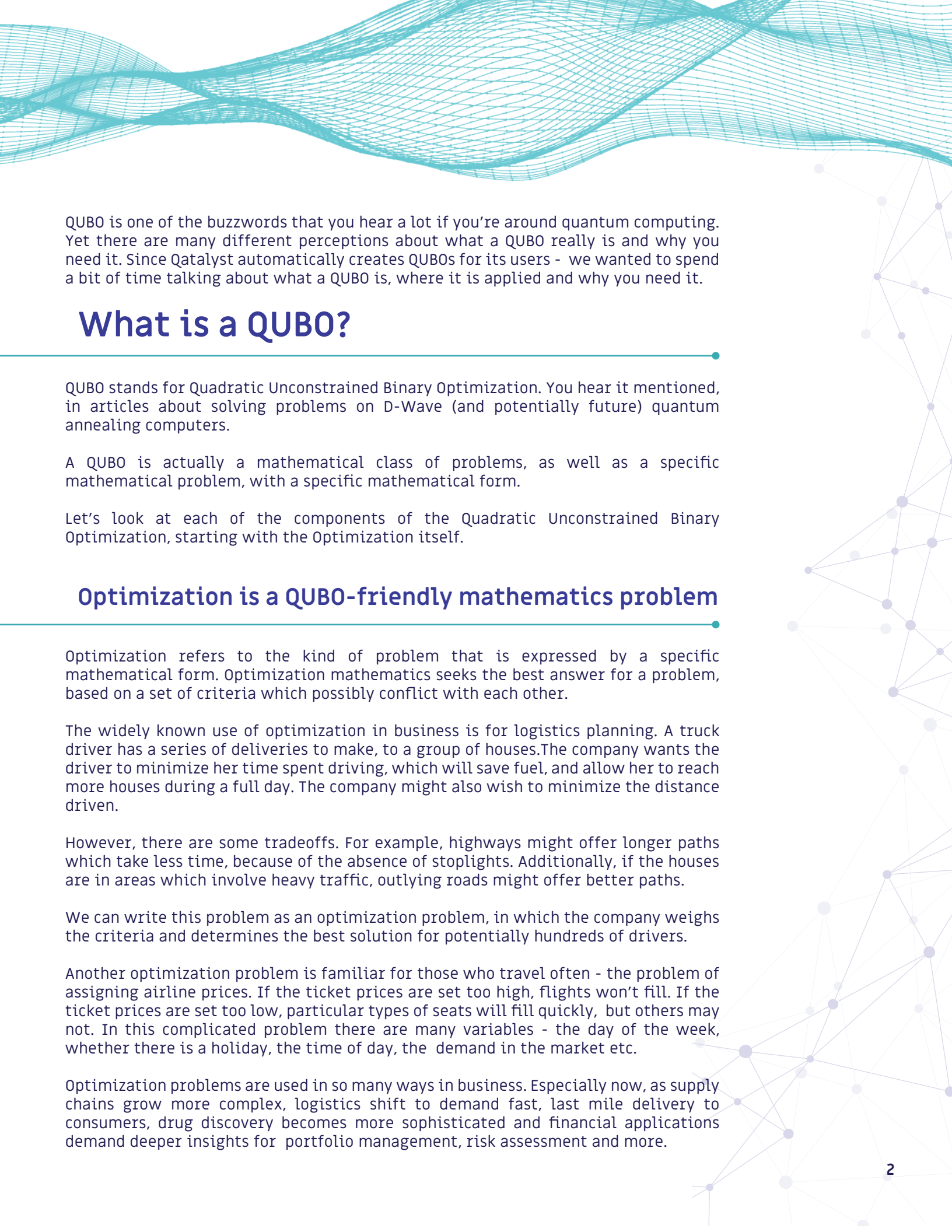
$$\hat{H} = \sum_{n=1}^N \frac{\hat{p}_n^2}{2m_n} + V(x_1, x_2, \dots, x_N)$$

$$E^2 - (pc)^2 = (mc^2)^2,$$

$$\Psi^*(x,t) \Psi(x,t)$$



Optimization with QUBOs: A Primer



QUBO is one of the buzzwords that you hear a lot if you're around quantum computing. Yet there are many different perceptions about what a QUBO really is and why you need it. Since Qatalyst automatically creates QUBOs for its users - we wanted to spend a bit of time talking about what a QUBO is, where it is applied and why you need it.

What is a QUBO?

QUBO stands for Quadratic Unconstrained Binary Optimization. You hear it mentioned, in articles about solving problems on D-Wave (and potentially future) quantum annealing computers.

A QUBO is actually a mathematical class of problems, as well as a specific mathematical problem, with a specific mathematical form.

Let's look at each of the components of the Quadratic Unconstrained Binary Optimization, starting with the Optimization itself.

Optimization is a QUBO-friendly mathematics problem

Optimization refers to the kind of problem that is expressed by a specific mathematical form. Optimization mathematics seeks the best answer for a problem, based on a set of criteria which possibly conflict with each other.

The widely known use of optimization in business is for logistics planning. A truck driver has a series of deliveries to make, to a group of houses. The company wants the driver to minimize her time spent driving, which will save fuel, and allow her to reach more houses during a full day. The company might also wish to minimize the distance driven.

However, there are some tradeoffs. For example, highways might offer longer paths which take less time, because of the absence of stoplights. Additionally, if the houses are in areas which involve heavy traffic, outlying roads might offer better paths.

We can write this problem as an optimization problem, in which the company weighs the criteria and determines the best solution for potentially hundreds of drivers.

Another optimization problem is familiar for those who travel often - the problem of assigning airline prices. If the ticket prices are set too high, flights won't fill. If the ticket prices are set too low, particular types of seats will fill quickly, but others may not. In this complicated problem there are many variables - the day of the week, whether there is a holiday, the time of day, the demand in the market etc.

Optimization problems are used in so many ways in business. Especially now, as supply chains grow more complex, logistics shift to demand fast, last mile delivery to consumers, drug discovery becomes more sophisticated and financial applications demand deeper insights for portfolio management, risk assessment and more.



Next up, Binary

A solver refers to the software, and sometimes hardware, that runs the computations to solve an optimization problem. We say that we send the problem to the solver, and the solver returns the best solutions it could find. It is then up to the user to interpret the solutions in terms of the problem's binary variables, to see whether the optimization criteria have been met successfully.

In the above case, binary means that the mathematical form is expressed in binary variables; that is, variables which can only take two values, "off" and "on." Often we write 0 for the value "Off" and 1 for the value "On," so that the binary variables are just 0 and 1. There are other choices; for example, in Ising model problems (named after the German physicist Ernst Ising), the binary choices are -1 and 1.

How can we solve a complicated problem by using variables which can only take two values?

First, let's look at what's called constrained optimization. An optimization problem is routinely expressed in terms of a set of variables, an objective function (a quantity to be minimized or maximized) and a set of constraints.

Consider choices for that delivery truck we mentioned earlier. Suppose it has four different routes between house A and house B. If we're expressing the problem in binary variables, we could use four binary variables. The first represents the decision: do we choose the first route? Yes or No? The second route represents the decision: do we choose the second route? Yes or No? You get the idea. You can also realize that when we actually try the optimization, we want only one of these binary variables to be Yes, while the others need to be No. This is called a constraint - a condition on the binary variables.

Confused about the Unconstrained in the QUBO?

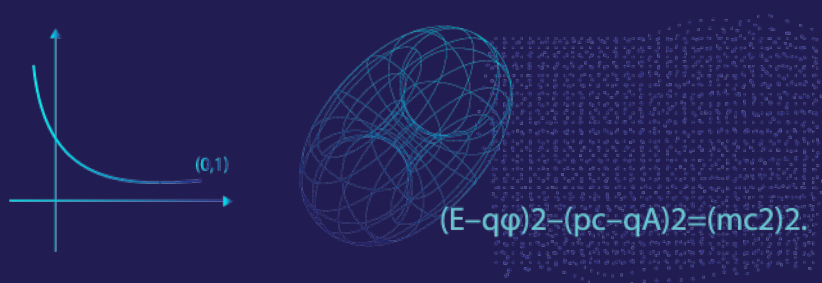
What does Unconstrained mean, given that we just explained what a constraint is?

The term Unconstrained in the QUBO definition refers to the fact that QUBOs allow the computation to satisfy all the constraints, but won't rigorously insist that all the constraints **MUST** be solved.

A QUBO gives us methods to penalize potential solutions which violate constraints, but makes it possible to still find solutions which violate constraints. In this sense, the QUBO is unconstrained because we have not rigorously insisted in following all of the constraints.

Consider the alternate routes to be taken, in the truck problem we've been discussing. If we can choose between four possible routes between A and B, we clearly have a constraint which should be followed by selecting one, and only one, of those routes. But we do not force the solutions to select only one. We allow the computation to find the best solutions - optimized solutions - which have only one of those routes selected.

This offers an advantage with QUBOs when processing optimizations, since it allows for multiple answers to be provided as results, even though they do not rigorously meet every single constraint. In large scale problems, this diversity of results often offers much deeper insights than a single, constrained result.



What about Quadratic?

Finally, let's consider the word quadratic. You might remember back to your high school algebra, where the word quadratic means "to the power two". You might remember graphs like circles and parabolas, which were equations involving quadratic powers of a single variable.

The word quadratic tells us that the mathematical form of a QUBO may not include anything besides linear terms (like x and y , if those are the independent variables), and xy (known as a "pairwise quadratic" term).

Problems expressed in terms of QUBOs are sometimes said to be "models," and the mathematics of a particular problem determines whether linear terms are sufficient, or whether quadratic terms are needed as well. For example, consider a model that attempts to predict the NCAA basketball tournament winners. Linear terms might describe an individual team's conference and competitive level of opponents in general; and linear terms might describe whether a team's roster is depleted, etc. In contrast, quadratic terms would add "pair terms" describing how one team performed versus another specific team, during the course of a season. The research involved in creating quadratic terms might be more expensive and detailed; and a betting pool competitor might want to use only linear terms to save money.

The Structure of a QUBO

There are many ways to write a QUBO. One common way is to express the QUBO's terms as a matrix. Another way to express the QUBO is to write the QUBO's terms in a file, and then read the file into a program. In this post, we'll express the QUBO algebraically. We'll start with the two types of terms, and then work an example.

Linear terms

A QUBO includes linear terms in each binary variable, which are composed of a constant (which is set by the programmer), and the binary variable. Examples of these are: $-2.7x$, $14y$, $0.2z$, where the binary variables are x , y and z .

Pairwise quadratic terms

A QUBO includes terms involving two binary variables, and a constant (which is set by the programmer). Examples of these are: $1.2xy$, $-3yz$, $0.4xz$, where the binary variables are again x , y and z .

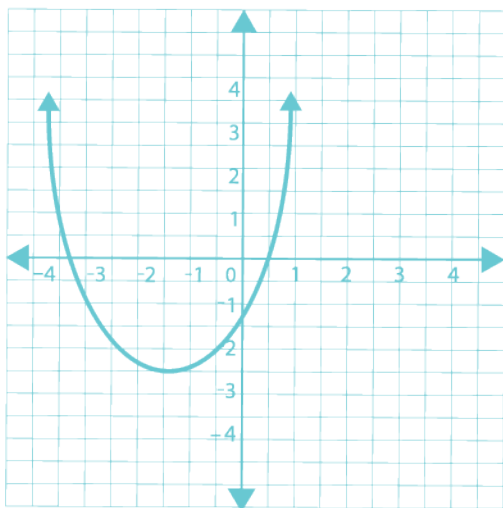
Where are the quadratic terms?

You may be wondering why there are not terms like x^2 , since QUBO is quadratic. The reason is that for binary variables 0 and 1, $x^2 = x$.

A Two-Variable QUBO

To sum up this section, here's the most general QUBO for two binary variables x and y :
 $\text{QUBO} = Ax + By + Cxy$

Now the question is - we mentioned earlier that the constants A , B and C are set by the programmer. How does the programmer choose those values?

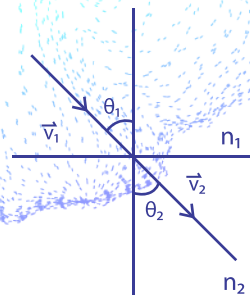


$$f(x) = a(x-x_1)(x-x_2)$$

$$f(x) \leq 5$$

$$x^2 - 4x + 5 \leq 5$$

$$x^2 - 4x \leq 0$$



$$\Psi^*(x,t)\Psi(x,t)$$



$$E^2 - (pc)^2 = (mc^2)^2,$$

Solving A Two-Variable Problem

Suppose that we want to solve a two-variable problem: 1) find the combinations for which the variables have the same value; 2) influence the solver so that it returns these two combinations with lower QUBO value than other combinations.

You might be wondering - this seems like such a small problem. How can it be important? It turns out that solving larger problems depends on understanding little problems like this.

First, for two binary variables, there are four possible combinations: 00, 01, 10 and 11. (In general, for N binary variables, there are 2^N combinations, so the algebra gets complicated quickly). Now we want the combinations for which the variables have the same value; that is clearly 00 and 11. The question has then become: How do we set the constants A and B and C so that the QUBO has a lower value for 00 and 11 than for 01 and 10?

First, notice that the QUBO expression will have zero value when we insert $x = 0$ and $y = 0$:

$$\text{QUBO} = A(0) + B(0) + C(0)(0) = 0$$

Noticing this, we realize that we also need to get a value of 0 for the combination 11, so that the two combinations with same variable values will have the same QUBO value. This means we need to have:

$$\text{QUBO} = A(1) + B(1) + C(1)(1) = 0$$

This doesn't tell us very much. However, we also know that the combinations 01 and 10 must have a higher QUBO value; and putting in 10 to the QUBO, we get:

$$\text{QUBO} = A(1) + B(0) + C(1)(0) = A$$

We penalize the 10 combination - give it higher QUBO value - by setting $A = 1$. (We have chosen 1 for simplicity - it could be any number larger than zero.) If we do the same thing with the 01 combination, we will get $B = 1$. Now, if we go back to the 11 combination, we will have:

$$\text{QUBO} = (1)(1) + (1)(1) + C(1)(1) = 0$$

and this indicates that we must have $C = -2$. Thus, our QUBO will be:

$$\text{QUBO} = x + y - 2xy$$

What do we do with this QUBO, now that we have it?

Sending the QUBO for Processing

We can send this QUBO to QCI's Qatalyst quantum software to be solved. We will need to follow this process:

1. Express the QUBO as linear and pairwise quadratic terms (we already did this)
2. Either write the QUBO values (the constants) into a computer program, or into a data file.
3. Include statements in the computer program which call the solver's API. The solver's API (for example, the Qatalyst API) will take care of sending the QUBO to the computation engine, and receiving the results which the engine returns. (known as the bitstring or solution)
4. Check that the solutions returned do solve the original problem you sent.

The Bottom Line

In the future, we expect developers and companies to build software tools which eliminate the need for some of the steps in this process. We started with this small problem, though, so that you would understand what needs to happen, under the surface.

Your assignment - if you choose to accept it - is to get access to a quantum computer and try it out!



Contact us if you're interested in accessing quantum computers using Qatalyst, our ready-to-run quantum software for quantum optimization.