frontiers in Physics

TECHNOLOGY REPORT
published: 14 June 2018
doi: 10.3389/fphy.2018.00055

Quantum-Assisted Cluster Analysis on a Quantum Annealing Device

Florian Neukart[1]*, *David Von Dollen*[1] and *Christian Seidel*[2]

[1] Volkswagen Group, Herndon, VA, United States, [2] Volkswagen Data:Lab, Munich, Germany

We present an algorithm for quantum-assisted cluster analysis that makes use of the topological properties of a D-Wave 2000Q quantum processing unit. Clustering is a form of unsupervised machine learning, where instances are organized into groups whose members share similarities. The assignments are, in contrast to classification, not known a priori, but generated by the algorithm. We explain how the problem can be expressed as a quadratic unconstrained binary optimization problem and show that the introduced quantum-assisted clustering algorithm is, regarding accuracy, equivalent to commonly used classical clustering algorithms. Quantum annealing algorithms belong to the class of metaheuristic tools, applicable for solving binary optimization problems. Hardware implementations of quantum annealing, such as the quantum annealing machines produced by D-Wave Systems [1], have been subject to multiple analyses in research, with the aim of characterizing the technology's usefulness for optimization, sampling, and clustering [2–17]. Our first and foremost aim is to explain how to represent and solve parts of these problems with the help of the QPU, and not to prove supremacy over every existing classical clustering algorithm.

Keywords: quantum computing, machine learning, quantum physics, clustering, quantum algorithms, quantum-assisted

OPEN ACCESS

Edited by:
Mark Everitt,
Loughborough University,
United Kingdom

Reviewed by:
George Siopsis,
University of Tennessee, Knoxville,
United States
Jonas Maziero,
Universidade Federal de Santa Maria,
Brazil

***Correspondence:**
Florian Neukart
florian.neukart@vw.com

Specialty section:
This article was submitted to Quantum Computing, a section of the journal Frontiers in Physics

Received: *07 March 2018*
Accepted: *17 May 2018*
Published: *14 June 2018*

Citation:
Neukart F, Dollen DV and Seidel C (2018) Quantum-Assisted Cluster Analysis on a Quantum Annealing Device. Front. Phys. 6:55. doi: 10.3389/fphy.2018.00055

INTRODUCTION

The idea behind quantum-assisted machine learning [18–26] is to either take parts of a classical algorithm and augment the tricky parts with a quantum subroutine or to find entirely new algorithms that exploit quantum effects and/ or the specific topology of a quantum processing unit (QPU). Most of the commercially available QPUs we see today are based on superconducting qubits, and couplers connecting these. The couplers can be seen as weighted connections and the qubits as vertices, so any problem that can exploit the topology of the QPU may be worth examining. The introduced quantum-assisted clustering algorithm falls into that category, as it utilizes the topological properties of the chip for assigning clusters. Besides the known problems, we see the potential for solving problems such as the finite element method, neural networks, and traffic flow, to name only a few, as in all of these we see interactions between elements and seek to minimize a quantity.

Quantum annealing is a class of algorithmic methods and metaheuristic tools for solving search or optimization problems. The search space for these problems usually consists of finding a minimum or maximum of a cost function. In searching a solution space for a problem, quantum annealing leverages quantum-mechanical superposition of states, where the system follows a time-dependent evolution, where the amplitudes of candidate states change in accordance of the strength of the transverse field, which allows for quantum tunneling between states. Following an adiabatic

process, a Hamiltonian is found whose ground state closely describes a solution to the problem [1, 2, 27].

Quantum annealing machines produced by D-Wave Systems leverage quantum annealing via its quantum processor or QPU. The QPU is designed to solve an Ising model, which is equivalent to solving quadratic unconstrained binary optimization (QUBO) problems, where each qubit represents a variable, and couplers between qubits represent the costs associated with qubit pairs. The QPU is a physical implementation of an undirected graph with qubits as vertices and couplers as edges between them. The functional form of the QUBO that the QPU is designed to minimize is:

$$Obj\,(x, Q) = x^T \cdot Q \cdot x \tag{1}$$

where x is a vector of binary variables of size N, and Q is an $N \times N$ real-valued matrix describing the relationship between the variables. Given the matrix Q, finding binary variable assignments to minimize the objective function in Equation (1) is equivalent to minimizing an Ising model, a known NP-hard problem [16, 28].

CLASSICAL CLUSTERING

In cluster analysis, the aim is to group sets of objects, i.e., points or vectors in d-dimensional space, such that some objects within one group can be clearly distinguished from objects in another group. An additional task may be the ability to quickly assign new objects to existing groups (clusters), i.e., by calculating the distance to a previously calculated cluster-centroid instead of running the re-running the complete clustering algorithm.

Clustering is a form of unsupervised machine learning, and used to find representative cases within a data set for supporting data reduction, or when needing to identify data not belonging to any of the found clusters [29]. Clustering helps to identify instances similar to one another, and to assign similar instances to a candidate cluster. A set of clusters is considered to be of high quality if the similarity between clusters is low, yet the similarity of instances within a cluster is high [30]. The groups are, in contrary to classification, not known a priori, but produced by the respective clustering algorithm [31]. Clustering is, amongst others, supported by self-organizing feature maps, centroid-based algorithms [32], distribution-based algorithms, density-based algorithms, orthogonal partitioning clustering.

We only explain one very common algorithm in detail—self-organizing feature maps—as this classical algorithm shares some similarities to the introduced quantum-assisted clustering algorithm.

Self-Organizing Feature Map

Self-organizing feature maps (SOFMs) are used to project high-dimensional data onto a low-dimensional map while trying preserve the neighboring structure of data. This means that data close in distance in an n-dimensional space should also stay close in distance in the low-dimensional map—the neighboring structure is kept. SOFMs inventor, Teuvo Kohonen, was inspired by the sensory and motor parts of the human brain [33].

Figure 1—Self organizing feature map—the scheme of a SOFM shows that every component of the input vector x is represented by an input neuron and is connected with the low dimensional layer (in this case) above it. During a learning phase, the weight vectors of a SOFM are adapted by self-organization [34]. As other artificial neural networks (ANNs), the SOFM consists of neurons $(n_1, \ldots, n_n)$, each having a weight vector w_i and a distance to a neighbor neuron. The distance between the neurons n_i and n_j is n_{ij},. As **Figure 1**—shows, each neuron is allocated a position in the low-dimensional map space. As in all other ANNs, initially the neuron weights are randomized. During learning, the similarity of each input vector to the weights of all neurons on the map is calculated, meaning that each member of the set of all weight vectors D is compared with the input vector $d \in D$. The SOFMs learning algorithm therefore belongs to the group of unsupervised learning algorithms. The neuron showing the highest similarity, having the smallest distance d_{small} to $d \in D$ is then selected as the winning neuron n_{win} (Equation 2) [35]:

$$d_{small} = \min_{1 \le j \le n} d\{(d \in D, w_j)\} \tag{2}$$

The weights of the winning neuron are adapted, as well as the weights of the neighbor neurons utilizing the neighborhood function φ_n and the learning rate μ. The neighborhood function has the following characteristics [35]:

- μ has its center at the position of n_{win} and is a maximum at this point.
- The neighboring neurons are considered according to a radius. Within this radius, for distances smaller than r, φ_n leads to outcomes greater than zero, and for distances greater than r, it takes on a value of zero.

Choosing a Gaussian function fulfills all the requirements in this case. The adaption of the weights is then carried out as described in Equation (3):

$$w_i^{(t+1)} = w_i^{(t)} + \mu\varphi_n \left(w_{n_{win}}, w_i^{(t)}, r \right) \left(d \in D - w_i^{(t)} \right) \tag{3}$$

During training, the learning rate and the neighborhood radius has to be reduced in each iteration, done by $\sigma^{(t+1)}$ (Eq. 4) [35, 36]:

$$\sigma^{(t+1)} = \sigma_s^* \left(\frac{\sigma_e}{\sigma_s} \right)^{(t+1)/(t+1)_e} \tag{4}$$

where σ_s represents the starting value and σ_e the ending value, also being the function value of $t(+1)_e$.

Similarities to SOFM and Quantum-Assisted Clustering

In the example depicted in **Figure 1**, the SOFM is a two-dimensional lattice of nodes, and depending on a presented instance, different nodes will fire with different strengths. The ones firing with the greatest amplitude give the cluster assignment. The QACA works similar in the sense that the two-dimensional topological properties of the D-Wave are exploited for cluster assignments. Assuming we embed two-dimensional clusters on the chip (higher-dimensional structures

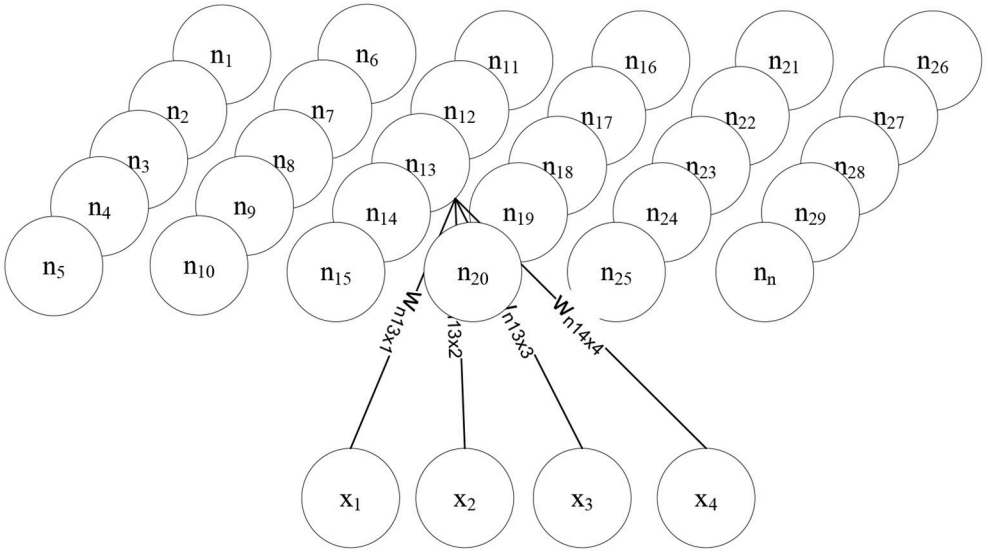

FIGURE 1 | Self-organizing feature map.

can be mapped as well—see the explanations in chapter 3), an assignment of cluster points to qubits may look as described in **Figure 2**:

Figure 2 shows schematically that qubits 1–8, and 17, 18, 21 would "fire," thus take the value 1 in the result-vector, and qubits 9–16 and 19, 20, 22–24 would not fire, thus take the value 0. We need to set the couplings accordingly, so that when a candidate instance is fed into the cluster-form (see QUBO-form and embedding, **Figure 3**) and embedded onto the QPU, the result allows us identify "areas" of activity or groups of qubits set to 1 for similar instances.

Figure 3 shows how an instance is fed into the cluster-form. $\overrightarrow{X} = (x_1, \ldots, x_n)$ represents the input vector.

QUANTUM-ASSISTED CLUSTERING ANALYSIS (QACA)

The introduced algorithm can be used as a probabilistic and definite clustering-algorithm, depending on how the result-vector is interpreted.

Quantum-Assisted Clustering With *n*-Dimensional Polytypes

The underlying idea is to classically define n-dimensional polytypes, such as the tetrahedron, the pentachoron, the tesseract, or even typeless polygons, which serve as clusters into which the instances projected, and map these onto the two-dimensional graph of the quantum annealing chip. The structure is derived from the number of input attributes in the data set. If each instance comes with 3 attributes, the structure of choice is a tetrahedron, and if the number of input attributes is 4, the structure of choice is a tesseract (see **Figure 4**).

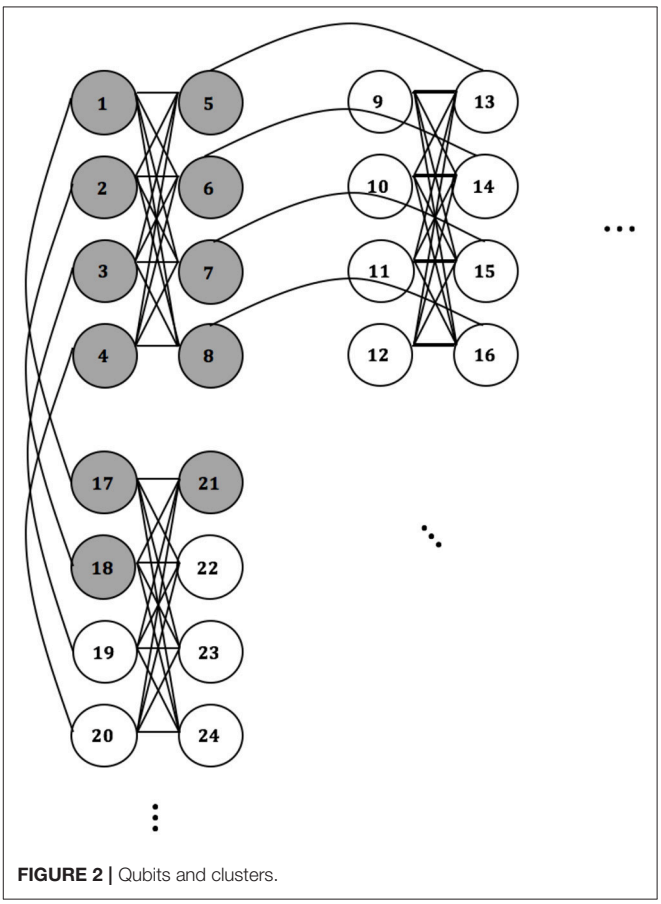

FIGURE 2 | Qubits and clusters.

The tetrahedron in three dimensions is given by four vertices, and assuming the intention is to cluster a four-dimensional data set into three clusters, three tetrahedra need to be defined. We

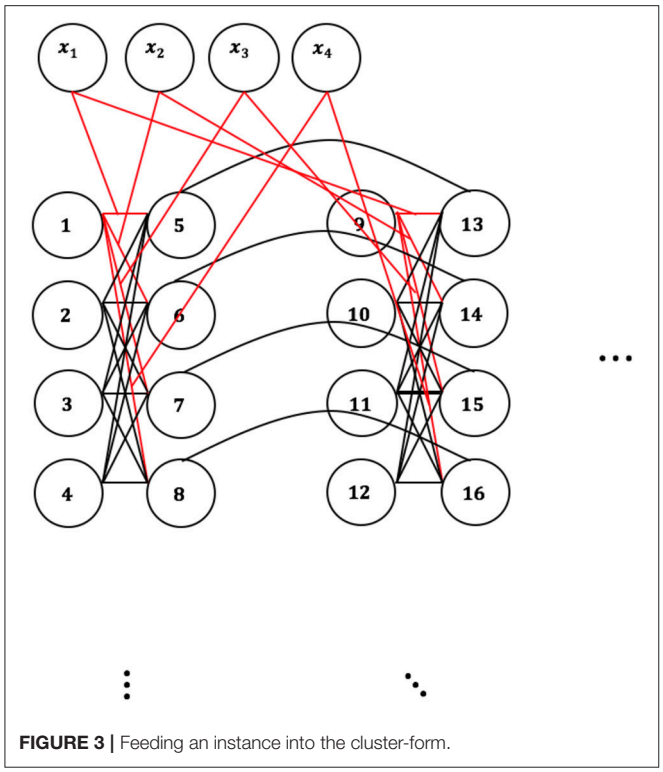

FIGURE 3 | Feeding an instance into the cluster-form.

do this by generating three random centroids, from which we calculate the remaining vertices. The centroid of the tetrahedron in **Figure 4** is given by the coordinates $c = (2, 2, 2)$. The remaining coordinates can be easily calculated, depending on the desired cluster size. Assuming we define a distance of 2 from the centroid, the set of tetrahedral coordinates $P = \{p_1, p_2, p_3, p_4\}$ are calculated as described in Equations 5–9:

$$p_1 = (c_x, c_y, c_z + 2) \tag{5}$$
$$p_2 = (c_x - 2, c_y - 2, c_z - 2) \tag{6}$$
$$p_3 = (c_x + 2, c_y - 2, c_z - 2) \tag{7}$$
$$p_4 = (c_x, c_y + 2, c_z - 2) \tag{8}$$

where the centroid c is defined as

$$c = (c_x, c_y, c_z) \tag{9}$$

As this approach does not generalize to other polytypes, the three-dimensional tetrahedron serves only as an example. Another way of defining clusters is by typeless polygons, based on randomly chosen coordinates from within a range of $min(x)$ and $max(x)$. Due to the inner workings of the introduced algorithm strongly overlapping clusters can be seen as probabilistic clustering, and clusters within clusters would help to identify clusters in data sets such as described in **Figure 5**, where the rows describe different the data sets, and the columns some algorithms used to cluster them:

Depending on how far we move the clusters apart, the less probabilistic QACA becomes, as the farther the clusters are apart, the smaller the probability of overlapping clusters becomes. If, classically (non-quantum), clusters do not overlap at all, we find definite cluster assignments for each of the instances. To give a first indication about how we define probability in terms of the introduced quantum-assisted clustering algorithm, we consider definite states of qubits post-measurement. Each qubit can be in one of the states $S = \{-1, 1\}$. The more qubits of a cluster $k_x \in K = \{k_0, \ldots, k_{m-1}\}$ take the state 1 for a specific instance $i_x \in I = \{i_0, \ldots, i_{l-1}\}$, the more probable it is that the instance i_x is a member of k_x. What's particularly elegant about this approach is that if clusters do not overlap in space, the nature of our algorithm still allows for probabilistic clustering (and to solve non-linear problems as depicted in **Figure 3**). However, the farther apart we move the clusters, the more the respective cluster coordinates differ from each other, and the more likely it is that we find definite assignments. We initialize the clusters based on n-dimensional typeless polygons as described in Algorithm 1:

Algorithm 1 Cluster definition based on n-dimensional typeless polygons.

Initialize: $\mathbf{i_c}$, $\mathbf{n_v}$, $\mathbf{M}$, $\mathbf{i_+}$, $\mathbf{r_{min}}$, $\mathbf{r_{max}}$
  For each k ∈ M:
    For each v ∈$\mathbf{N_v}$:
      $\mathbf{v_x^c}$= **rand** ($\mathbf{r_{min}}$,$\mathbf{r_{max}}$)
      $\mathbf{v_y^c}$= **rand** ($\mathbf{r_{min}}$,$\mathbf{r_{max}}$)
      $\mathbf{v_z^c}$= **rand** ($\mathbf{r_{min}}$,$\mathbf{r_{max}}$)
    $\mathbf{r_{min}}$=$\mathbf{r_{min}}$+$\mathbf{i_+}$∗ϵ
    $\mathbf{r_{max}}$=$\mathbf{r_{max}}$+$\mathbf{i_+}$∗ϵ
  Breakdown
  $\mathbf{i_c}$: the initial coordinate for cluster vertex calculations, given by Eq. 8.
  $\mathbf{n_v}$: set of all vertices per cluster, i.e., four vertices per cluster: $\mathbf{N_v} = \{1, 2, 3, 4\}$.
  k: cluster
  M: set of all clusters, i.e., three clusters: $\mathbf{M} = \{1, 2, 3\}$.
  $\mathbf{i_+}$: increment by which the coordinate range for finding random vertices is shifted, given by Eq. 9.
  $\mathbf{r_{min}}$: minimum range value for finding random vertices which define a cluster. Initialized as $\mathbf{r_{min}}$=$\mathbf{i_c}$.
  $\mathbf{r_{max}}$: maximum range value for finding random vertices which define a cluster. Initialized as $\mathbf{r_{max}}$=$\mathbf{r_{min}}$+$\mathbf{i_+}$.
  $\mathbf{v_x^c}$, $\mathbf{v_y^c}$, $\mathbf{v_z^c}$: x, y, z coordinates of the vertex v in the c^{th} cluster. In the introduced example space is 3-dimensional, but the algorithm generalizes to n-dimensional space, and even complex manifolds.
  ϵ: sliding factor.

$$i_c = min(X) \tag{10}$$
$$i_+ = \frac{\max(X) - min(X)}{m} \tag{11}$$

where X is the matrix of input attributes and m the number of clusters. In Algoritham 1, we assign coordinates to each vertex of an n-dimensional typeless polygon. For each cluster,

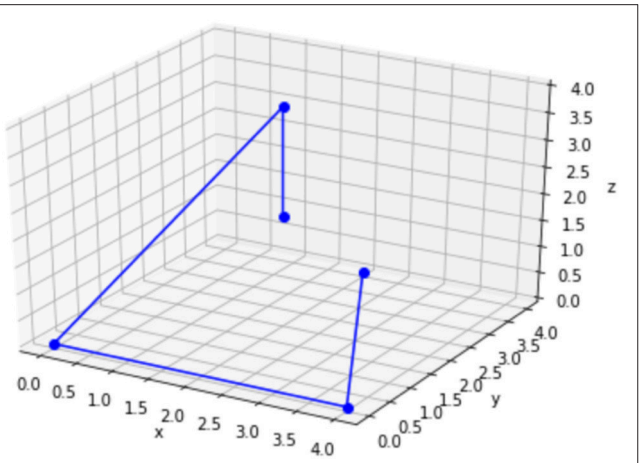

FIGURE 4 | Tetrahedron in three dimensions.

we shift the coordinate range $r = (r_{min}, r_{max})$ by the increment i_+ and a sliding factor i_+, which is increases or decreases in coordination with desired inter-cluster distances. We emphasize that large inter-cluster distances, i.e., in the Euclidean sense, do not necessarily imply definite cluster assignments. For an instance i_x, the introduced algorithm may still calculate a certain probability of i_x belonging to cluster k_1, but also to k_x, even when k_1 and k_x do not overlap in n-dimensional space.

QUBO-form and Embedding

We present the problem to the D-Wave in QUBO-form. The definition of the matrix in QUBO-form is done in two steps.

1. The first step is in defining a matrix in QUBO-form or what we call a cluster-form (CF). The CF is defined only once for all presented instances, and subsequently modified as instances are fed into it. It is worth pointing out another major difference to classical clustering algorithms such as k-means or self-organizing feature maps: instead of training regimes, i.e., iterative distance-based calculation of centroids, or strengthening the weights of nearest neighbors around a firing neuron, we only need to allocate instances to the CF once to obtain the cluster assignment.

   The QUBO-matrix is an upper triangular $N \times N$-matrix defined by $i \in \{0, \ldots, N-1\}$ by $j \in \{0, \ldots, N-1\}$. In the demonstrated example, each entry is initialized with 0, and subsequently updated with the values calculated for the CF, which come from Algorithm 1. The CF will hold all values of the vertices based on the simple calculations in Algorithm 1. While calculating each vertex coordinate v_x^c, v_y^c, v_z^c, we also assign an ID to each of these and store this information in a lookup-table. The x-coordinate in first vertex in the first cluster is given the ID 1: v_x^1 (or more accurately: $v_{1_x}^1$, where the exponent defines the cluster, and the subscript the vertex number and the respective coordinate), the y-coordinate in the first vertex of the first cluster the ID 2, and so on. We additionally create a list L of length $l = n_v * m$, which contains a list of the coordinate values, i.e., the first three entries of this list give the x, y, z coordinates of the first vertex in the first cluster. The values in L may also be scaled as described in Eq. 20, but this strongly depends from the variance in the data set. We define the number of vertices as n_v and m the number of clusters. Additionally, we store the qubit-to-cluster assignments in a lookup-table D in the form $\{k_1 : [0, 1, 2], k_2 : [3, 4, 5], \ldots, k_n : [q_{x-3}, \ldots, q_{x-1}]\}$ that we use in step 2. We assign k_x as the cluster number, and qubits are given by the respective arrays. The CF is then defined as described in Equation (12):

$$CF(i,j) = \begin{cases} CF(i,j) - \sqrt{\left(L_i^2 + L_j^2\right)}, & if c1 \\ CF(i,j) + \sqrt{\left(L_i^2 + L_j^2\right)}, & if\ c2 \\ CF(i,j), & otherwise \end{cases} \tag{12}$$

where

$$c1 : S_1 \equiv S_2\ and\ i \leq j \tag{13}$$

and

$$c2 : S_1 \neg \equiv S_2\ and\ i \leq j \tag{14}$$

In Equations (13, 14) the conditions for assigning positive or negative signs to an entry are defined. If c1 is met, our tests show that setting the respective entries to 0 instead of $-\sqrt{\left(L_i^2 + L_j^2\right)}$ may provide better results, but there is a noticeable variance over differing data sets. The basic idea is to iterate over the qubit-IDs of each cluster, and to compare if the set of qubit IDs S_1 is identical to the set of qubit IDs S_2. If the sets are identical, negative intra-cluster couplings are set, and if not, positive inter-cluster couplings are set. The reason for this is that once we introduce an instance to the CF. The coupling-strengths values around the most probable cluster's qubits are lowered, and in the same instance the values the inter-cluster couplings help to raise the entries of the remaining clusters. This results in lower probability of the most probable clusters being activated.

2. The second step is iterating over all cluster-instances: the instances are fed into the cluster-form one by one, and each of the resulting instance-cluster matrices (ICM) are embedded on the QPU. For each cluster, we go over the number of vertices and calculate a distance from each attribute-coordinate to each cluster-coordinate. The number of qubits per cluster must be a multiple of the number of data set attributes, i.e., when the data set is three-dimensional, a cluster may be represented by 3 qubits (point), 6 qubits (line), 9 qubits(triangle), and so on. If a cluster in a 3-dimensional space is defined by 6 points, we require 18 qubits to represent it on the QPU. For each of the cluster coordinates, we now calculate the distance to each instance and update the list L accordingly. L, as defined in step 1, was used to define the cluster-form and was set with negative intra-cluster couplings, and positive inter-cluster couplings. For each instance, L is updated as described in Algorithm 2:

Algorithm 2 Instance to cluster distance calculation.

Load: **D**, **L**, $\mathbf{i_x}$
Initialize: ***cc=0***
  For each k ∈ D:
    For each qubit ∈ k:
      $L\left[qubit\right] = L\left[qubit\right] - i[cc]^2$
      $cc = cc + 1$
      if $cc == d$: **then**
        cc=0
      end if
  Breakdown
D: Cluster dictionary $D: \left\{k_1 : [0, 1, 2]\,,\ k_2 : [3, 4, 5]\ , \ldots,\ k_n : \left[q_{x-3}, \ldots,\ q_{x-1}\right]\right\}$
L: List with qubit-IDs and their values as initialized in the cluster-form
$\mathbf{i_x}$: an instance
cc: coordinate counter. Counts up to 3 if the instance has 3 coordinates, up to 4 with 4 coordinates, and so on
d: number of dimensions per instance
k: key/ cluster in *D*
qubit: the qubit IDs per entry in *D*
L[**qubit**]: the value of *L* at entry *qubit*

With Algorithm 2, the distance from an instance i_x to any point in any cluster in the cluster-form is calculated. Once this is done, the ICM is updated as described in Equations 15–20:

$$CF\left(i,j\right) = \begin{cases} CF\left(i,j\right) - \left(L_i^2 + L_j^2\right), & if c1 \\ CF\left(i,j\right) - \left(L_i^* L_j\right), & if\ c2 \\ CF\left(i,j\right) + \left(L_i^2 + L_j^2\right), & if\ c3 \\ CF\left(i,j\right) + \left(L_i^* L_j\right), & if\ c4 \\ CF\left(i,j\right), & otherwise \end{cases} \tag{15}$$

where

$$c1:\ S_1 \equiv S_2\ \ and\ i < j \tag{16}$$

and

$$c2:\ S_1 \equiv S_2\ \ and\ i = j \tag{17}$$

and

$$c3:\ S_1 \neg \equiv S_2\ and\ \ i < j \tag{18}$$

and

$$c3:\ S_1 \neg \equiv S_2\ and\ \ i = j \tag{19}$$

The last step before embedding the problem onto the QPU is scaling the values in the ICM, which is done according to Equation (20):

$$x_{scaled} = \frac{x_i - mean(x)}{\sigma\ (x)} \tag{20}$$

where $\sigma\ (x)$ is the standard deviation. The features are centered to the mean and scaled to unit variance.

Once the ICM has been processed, the spin-directions provided in the result-vector tell us which qubits are "turned on," and which are "turned off." Three ways to extract the cluster assignments are probabilistic and definite:

1. Definite: For the turned-on qubits, the respective values of *L* are extracted, and by looking up *D* we can identify the cluster this qubit belongs to. In *D*, we can find the qubits per cluster, and from the result-vector we get the turned-on. We look up the respective IDs in *L*, and sum the values over the remaining qubits. The lowest sum of "on"-qubit values per cluster gives the cluster assignment.
2. Probabilistic 1: The number of turned-on qubits per cluster, as defined by qubit-assignments in *D*, is counted. The percentage of turned-on qubits per cluster gives the probabilistic assignments of an instance to clusters.
3. Probabilistic 2: For the turned-on qubits, the respective values of *L* are extracted, and by looking up *D* we can identify the cluster this qubit belongs to. In *D*, we can find the qubits per cluster, and from the result-vector we get the turned-on. We look up the respective IDs in *L*, and sum the values over the remaining qubits. The percentage of turned-on qubits-values per cluster gives the probabilistic assignments of an instance to clusters.

EXPERIMENTAL RESULTS AND CONCLUSIONS

Our intention was to obtain the results without having to split the QUBO so that a singular embedding is possible. We verified QACA with commonly used low-dimensional verification data sets, such as the Iris data set. For verification, we chose Expectation Maximization, k-means, and Self-Organizing Feature Maps, all three known to perform well on the Iris data set. We ran QACA 5 times and averaged the performance, as due to the randomness in the cluster-form the results can vary. In brackets, we provide the individual cluster assignments. The accuracy is defined as percentage of correctly assigned instances, and the cluster-assignment is definite (Tbl. 1).

Some example results for the "Probabilistic 2"-method, which is as accurate as the definite results described in **Table 1** when assigning highest probability to an instance, are as follows (**Table 2**):

Summing up, the quantum-assisted clustering algorithm can compete with classical algorithms in terms of accuracy, and sometimes outperforms the ones used for comparison on the test data sets. However, the results strongly vary depending on the cluster-form, and better ways for cluster-form initialization have to be found.

In the experiments described above, the number of used physical qubits used for the embedding varied from instance to instance, with a minimum of 967 and a maximum of 1,455. Finding a valid performance-comparison in terms of runtime is challenging due to the following reasons:

1. QACA does not require training, classical algorithms do: Compared to classical algorithms, the introduced algorithm does not require training, whereas classical algorithms' training time increases with the number of attributes and instances.

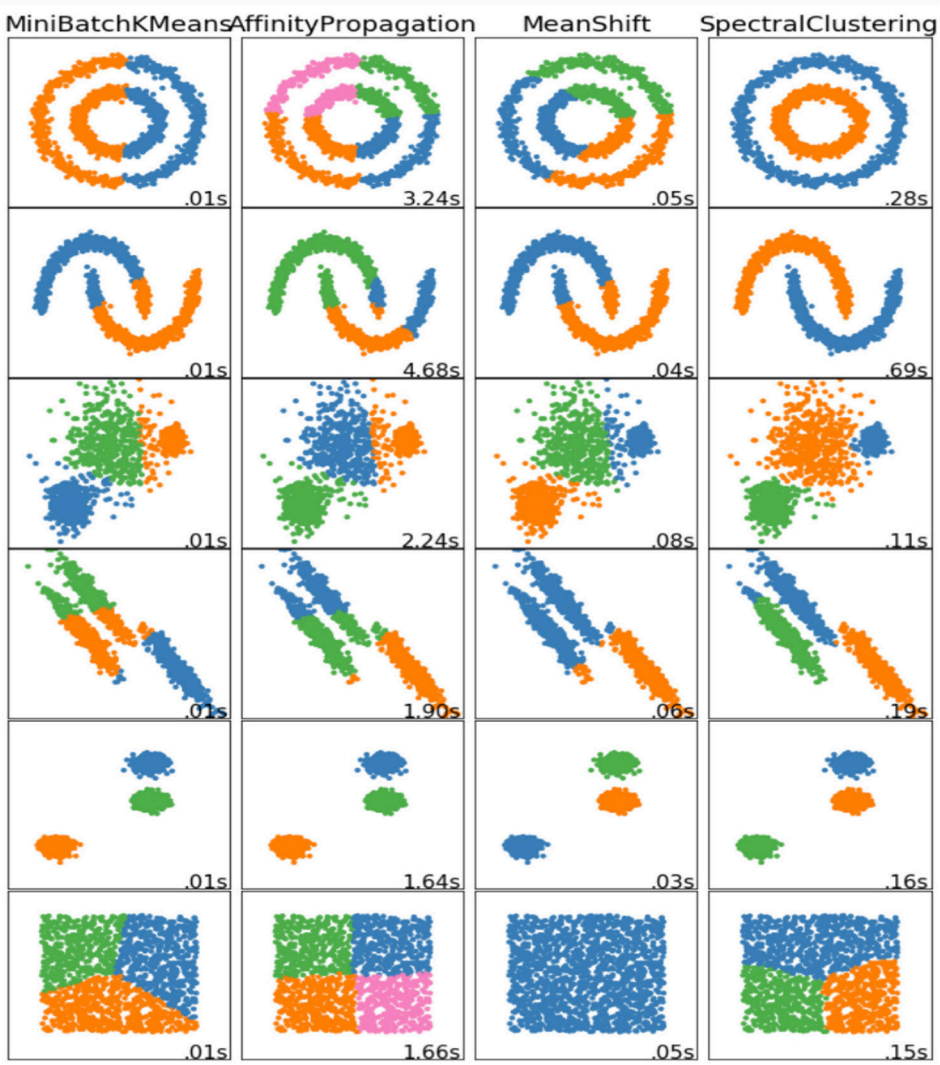

FIGURE 5 | Some non-linear data sets and some ways to cluster them [37].

TABLE 1 | Algorithm comparison.

	EM	k-means	SOFM	QACA
Accuracy in %	86	89.7	70.7	Avg.: ∼85.6 Ind.: (87.33 (131), 90 (135), 83.33 (125), 80 (120), 87.33 (131))

TABLE 2 | Probabilistic assignments.

instance 0 probabilities: 1.06, 20.96, 77.97
instance 1 probabilities: 1.06, 20.96, 77.97
instance 2 probabilities: 2.62, 20.99, 76.38
instance 3 probabilities: 0.76, 20.92, 79.83
instance 4 probabilities: 1.06, 20.96, 77.97
instance 5 probabilities: 4.019, 23.99, 80.02

2. QACA requires to embed instances, but classical algorithms don't: Classical algorithms will most likely always be faster in the application phase, as finding a solution on the D-Wave depends on the embedding time, which may vary from instance to instance.

Apart from the accuracy results, we see another strength of the QACA-algorithm in the ability to identify new clusters without training: if the introduced algorithm is not constrained by n predefined polytypes, but instances are fed into the whole lattice, different activation patterns allow the identification of unknown behavior.

FUTURE WORK

In our future work, we intend to further exploit the chip topology to identify cluster assignments. By identifying where on the QPU

we can find the turned-on qubits, an implementation of full feature map should be possible.

AUTHOR CONTRIBUTIONS

FN: problem formulation, code, and publication. DV and CS: publication.

ACKNOWLEDGMENTS

Thanks go to VW Group CIO Martin Hofmann and VW Group Region Americas CIO Abdallah Shanti, who enable our research. Special thanks go to Sheir Yarkoni of D-Wave systems whose valuable feedback helped us to present our results comprehensibly.

REFERENCES

1. Wave D. *Quantum Computing, How D-Wave Systems Work* [04-24-2017]. (2017). Available online at: https://www.dwavesys.com/quantum-computing
2. Benedetti M, Gmez J-R, Biswas R, Perdomo-Ortiz A. Estimation of effective temperatures in quantum annealers for sampling applications: a case study with possible applications in deep learning. *Phys Rev A*. (2015) **94**:022308. doi: 10.1103/PhysRevA.94.022308
3. Smelyanskiy VN, Venturelli D, Perdomo-Ortiz A, Knysh S, Dykman MI. Quantum annealing via environment-mediated quantum diffusion. *Phys Rev Lett*. (2015) **118**:066802. doi: 10.1103/PhysRevLett.118.066802
4. Venturelli D, Marchand DJJ, Rojo G. Quantum Annealing Implementation of Job-Shop Scheduling. arXiv: 1506.08479v2 [quant-ph] (2015).
5. Jiang Z, Rieffel EG, Non-commuting two-local Hamiltonians for quantum error suppression. *Quantum Inf Process* (2015)**16**:89. doi: 10.1007/s11128-017-1527-9
6. Isakov SV, Mazzola G, Smelyanskiy VN, Jiang Z, Boixo S, Neven H, et al. Understanding Quantum Tunneling through Quantum Monte Carlo Simulations. *Phys Rev Lett*. (2015) **117**:180402. doi: 10.1103/PhysRevLett.117.180402
7. O'Gorman B, Perdomo-Ortiz A, Babbush R, Aspuru-Guzik A, Smelyanskiy V. Bayesian network structure learning using quantum annealing. *Eur Phys J Spec Top*. (2015) **224**:163. doi: 10.1140/epjst/e2015-02349-9
8. Rieffel EG, Venturelli D, O'Gorman B, Do MB, Prystay E, Smelyanskiy VN. A case study in programming a quantum annealer for hard operational planning problems. *Quantum Inf Process* (2014) **14**:1.
9. Venturelli D, Mandr S, Knysh S, O'Gorman B, Biswas R, Smelyanskiy V. Quantum optimization of fully-connected spin glasses. *Phys. Rev*. (2014) **X5**:031040. doi: 10.1103/PhysRevX.5.031040
10. Perdomo-Ortiz A, Fluegemann J, Narasimhan S, Biswas R, Smelyanskiy VN. A quantum annealing approach for fault detection and diagnosis of graph-based systems. *Eur Phys J Spec Top*. (2014) **224**:131. doi: 10.1140/epjst/e2015-02347-y
11. Boixo S, Ronnow TF, Isakov SV, Wang Z, Wecker D, Lidar DA, et al. Evidence for quantum annealing with more than one hundred qubits. *Nat Phys*. (2014) **10**:218–24. doi: 10.1038/nphys2900
12. Babbush R, Perdomo-Ortiz A, O'Gorman B, Macready W, Aspuru-Guzik A. Construction of Energy Functions for Lattice Heteropolymer Models: efficient Encodings for Constraint Satisfaction Programming and Quantum Annealing Advances in Chemical Physics. arXiv:1211.3422v2 [quant-ph] (2012).
13. Smolin JA, Smith G. Classical signature of quantum annealing. *Front. Phys*. (2013) **2**:52. doi: 10.3389/fphy.2014.00052
14. Perdomo-Ortiz A, Dickson N, Drew-Brook M, Rose G, Aspuru-Guzik A. Finding low-energy conformations of lattice protein models by quantum annealing. *Sci Rep*. (2012) **2**:571. doi: 10.1038/srep00571
15. Neukart F, Von Dollen D, Seidel C, Compostella G. Quantum-Enhanced reinforcement learning for finite-episode games with discrete state spaces. *Front Phys*. (2018) **5**:71. doi: 10.3389/fphy.2017.00071
16. Los Alamos National Laboratory. *D-Wave 2X Quantum Computer*. (2016). Available online at: http://www.lanl.gov/projects/national-security-education-center/information-sciencetechnology/dwave/
17. Von Dollen D. identifying similarities in epileptic patients for drug resistance prediction. arXiv:1704.08361 (2017).
18. Springer Professional. *Volkswagen Trials Quantum Computers* (2017). Available online at: https://www.springerprofessional.de/en/automotive-electronics—software/companies—institutions/volkswagen-trials-quantum-computers/12170146?wtmc=offsi.emag.mtz-worldwide.rssnews.-.x
19. Neukart F. Quantum physics and the biological brain. In: AutoUni Schriftenreihe editor *Reverse Engineering the Mind*, Vol 94, Wiesbaden: Springer (2017). p. 221–229.
20. Lanting T, Przybysz AJ, Yu A, Smirnov FM, Spedalieri MH, Amin AJ, et al. Entanglement in a quantum annealing processor. *Phys. Rev. X*. (2013) **4**:021041. doi: 10.1103/PhysRevX.4.021041
21. Kumar V, Bass G, Tomlin C, Dulny J. III. Quantum annealing for combinatorial clustering. arXiv:1708.05753v2 (2018).
22. Lucas A. Ising formulations of many NP problems. *Front Phys*. (2014) **2**:5. doi: 10.3389/fphy.2014.00005
23. Neukart F, Moraru SM. Operations on quantum physical artificial neural structures. *Proced Eng*. (2014) **69**:1509–17. doi: 10.1016/j.proeng.2014.03.148
24. Korenkevych D, Xue Y, Bian Z, Chudak F, Macready WG, Rolfe J, et al. Benchmarking Quantum Hardware for Training of Fully Visible Boltzmann Machines. arXiv:1611.04528v1 [quant-ph] (2016).
25. Neukart F, Moraru SM. On quantum computers and artificial neural networks. *Signal Process Res*. (2013) **2**:1–11.
26. Levit A, Crawford D, Ghadermarzy N, Oberoi JS, Zahedinejad E, Ronagh P. Free-Energy-based Reinforcement Learning Using a Quantum Processor. arXiv:1706.00074v1 [cs.LG] (2017).
27. Crawford D, Levit A, Ghadermarzy N, Oberoi JS, Ronagh P. Reinforcement learning using quantum boltzmann machines. arXiv:1612.05695 [quant-ph] (2016).
28. Neukart F, Seidel C, Compostella G, Von Dollen D, Yarkoni S, Parney B. Traffic flow optimization using a quantum annealer. *Front ICT* (2017) **4**:29. doi: 10.3389/fict.2017.00029
29. Finilla AB, Gomez MA, Sebenik C, and Doll JD. Quantum annealing: A new method for minimizing multidimensional functions. *Chem Phys Lett*. (1994) **219**:343.
30. Neukart F. An outline of artificial neural networks. In: AutoUni S. editor, *Reverse Engineering the Mind*. vol. 94. Wiesbaden: Springer (2017). 91–3.
31. Anderberg MR. *Cluster Analysis for Applications*. New York, NY: Academic Press Inc. (1973).
32. Chamoni P, Gluchowski P. *Analytische Informationssysteme: Business Intelligence- Technologien und –Anwendungen, 3rd Edn*. Berlin: Springer (2006).
33. MacQueen JB. Some methods for classification and analysis of multivariate observations. In: *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability,* Vol. 1 (Berkeley: University of California Press) (1967). 281–97.

34. Oracle. *O-Cluster: Scalable Clustering of Large High Dimensional Datasets [2018-02-28]*. Oracle Corporation. Available online at: https://docs.oracle.com/cd/B28359_01/datamine.111/b28129/algo_oc.htm
35. Ritter H, Martinez T, Schulten K. *Neuronale Netze. Eine Einführung in die Neuroinformatik selbstorganisierender Netzwerke*. Massachusetts, MA: Addison Wesley (1991).
36. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. scikit-learn: machine learning in python. *JMLR* (2011) **12**:2825–30.
37. Kramer O. *Computational Intelligence: Eine Einführung*. Berlin: Springer (2009).

Conflict of Interest Statement: FN and DV were employed by Volkswagen Group, Region Americas. CS was employed by Volkswagen Group, Germany.