Neural optimization for quantum architectures: graph embedding problems with Distance Encoder Networks

Neural optimization for quantum architectures: graph embedding problems with Distance Encoder Networks

Chiara Vercellino*†‡, Giacomo Vitali*†, Paolo Viviani*, Alberto Scionti*, Andrea Scarabosio*, Olivier Terzo*, Edoardo Giusto†, Bartolomeo Montrucchio†

*LINKS Foundation, Torino, Italy
†DAUIN, Politecnico di Torino, Torino, Italy
‡chiara.vercellino@linksfoundation.com

***Abstract*—Quantum machines are among the most promising technologies expected to provide significant improvements in the following years. However, bridging the gap between real-world applications and their implementation on quantum hardware is still a complicated task. One of the main challenges is to represent through *qubits* (*i.e.*, the basic units of quantum information) the problems of interest. According to the specific technology underlying the quantum machine, it is necessary to implement a proper representation strategy, generally referred to as *embedding*. This paper introduces a neural-enhanced optimization framework to solve the constrained unit disk problem, which arises in the context of qubits positioning for neutral atoms-based quantum hardware. The proposed approach involves a modified autoencoder model, *i.e.*, the Distances Encoder Network, and a custom loss, *i.e.*, the Embedding Loss Function, respectively, to compute Euclidean distances and model the optimization constraints. The core idea behind this design relies on the capability of neural networks to approximate non-linear transformations to make the Distances Encoder Network learn the spatial transformation that maps initial non-feasible solutions of the constrained unit disk problem into feasible ones. The proposed approach outperforms classical solvers, given fixed comparable computation times, and paves the way to address other optimization problems through a similar strategy.**

***Index Terms*—embedding, graphs, neural networks, neutral atoms, optimization**

I. Introduction

In recent years, quantum computers have garnered more and more interest, as they represent very auspicious tools to accelerate specific computations like material simulations [1]–[3], combinatorial optimization [4]–[6] etc. However, we are currently in the noisy intermediate-scale quantum (NISQ) era. Thus practical applications of canonical quantum algorithms (*e.g.*, Shor's or Grover's algorithms) are still unattainable because of technical limitations such as low qubits count, limited coherence time, and gate fidelity. To overcome these limitations, different approaches emerged, like hybrid quantum-classical algorithms.

In this regard, our paper introduces a novel approach to exploiting neural networks (NN) to optimize neutral atoms' quantum architectures [7]. More precisely, the proposed framework deals with combinatorial optimization problems embedding into the previously mentioned quantum machines. The embedding task from an optimization point of view is equivalent to solving a constrained unit disk graph (CUDG) problem. This optimization problem concerns finding a unit disk graph (UDG) [8] (see Def. 1) realization, given a generic graph. Contextualizing the optimization problem in a real-world quantum computing (QC) application brings additional constraints to a standard UDG problem, thus completing its definition.

Definition 1 (Unit disk graph): Consider n circles, with radius r, in the plane. The intersection graph [9] of these circles is a unit disk graph with n vertexes where each vertex corresponds to a circle centre and the vertexes share an edge only if the corresponding circles intersect.

A. Unit disk graphs in quantum applications

Unit disk graphs have been popularized, in real-world applications, by wireless communication [10]–[12]. However, with the emergence of quantum technologies, UDGs have also become of interest in the quantum field. In particular, quantum neutral atoms machines [7] rely on Rydberg atoms, positioned on a 2D/3D register, to represent *qubits*. From the interactions between neutral atoms, subject to the action of laser pulses, a spin Hamiltonian (*e.g.*, Ising) is retrieved. The evolution of the spin Hamiltonian is related to qubits (neutral atoms) that, once measured, assume one out of two possible quantum states (*i.e.*, the excited Rydberg state $|1\rangle$ or the ground state $|0\rangle$), hence the association with binary optimization variables. This capability of neutral atoms machines enables the mapping of a large set of NP-hard combinatorial optimization problems into the Ising Hamiltonian [13].

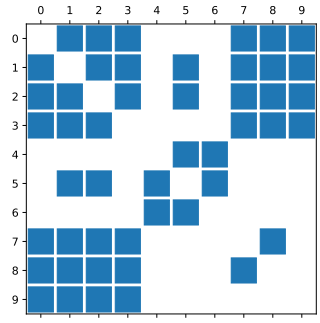
In principle, the class of problems that can benefit from this quantum-based solution paradigm [14], [15] are Quadratic Unconstrained Binary Optimization (QUBO) problems [16]. They are characterized by a square matrix $Q \in \mathbb{R}^{n \times n}$ and a vector of n binary variables $x \in \{0,1\}^n$. The complete definition of QUBO problems is obtained by minimizing the objective functions in the form $x^T Q x$. Theoretically, once the

qubits are organised in the space to reproduce the desired Hamiltonian, the associated QUBO problem can be solved using Quantum Approximate Optimization Algorithm [14], [15] or Quantum Adiabatic Algorithm [17].

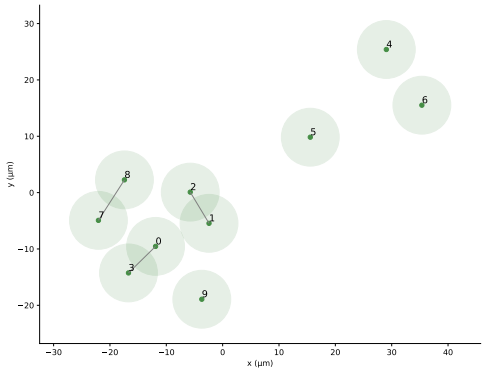
In neutral atoms' quantum architectures, the blockade effect is one of the main players. It is a threshold-like effect reached when the distances between qubit pairs are shorter than the blockade radius [18], *i.e.* a critical distance at which the strength of the interactions balances with the Rabi frequency of the laser pulses [19]. Therefore, the interactions between qubits in the register induce a UDG. In this situation, the halved blockade radius plays the role of the radius r, as in definition 1, and its appropriate value can be set according to the characteristics of the considered quantum device. The qubit positions instead correspond to the circle centres, so when n qubits are placed on the quantum register and are excited through laser pulses, the corresponding n-vertexes UDG can be retrieved.

However, the representation of the spin Hamiltonian for application use cases usually requires a backward approach: the off-diagonal elements of Q describe the connectivity pattern (*i.e.* the adjacency matrix) wanted in the quantum register. Atoms (*i.e.* qubits) are placed in 2D/3D configurations, and their interactions define UDGs, as reported in Fig 1b and Fig 1c. In sum, qubit positions are looked for so that the blockade effect reproduces a UDG that respects the desired connectivity.

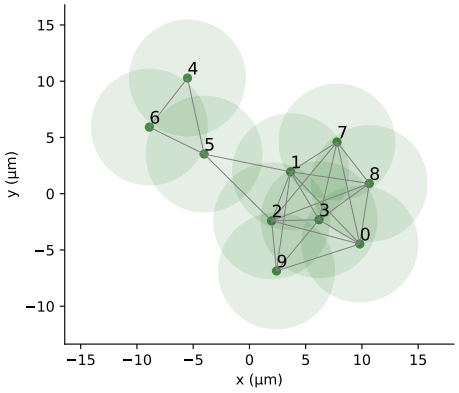
Moreover, other requirements come along with the specific quantum device, thus adding constraints to the UDG problem. For the hardware considered in this work, the tweezers governing qubits placement can not place atoms nearer than $D_{min} = 4\ \mu m$, the register can handle atoms placed within a circular area of radius $L = 50\ \mu m$, and the greatest value allowed for radius r is estimated at $D_{adj} \approx 10.26\ \mu m$. Finally, it is desirable that the qubits placement not only induces the wanted UDG configuration but also corresponds to a UDG solution that maximizes the *adjacency gap*. That means maximizing the difference between the minimum distance among qubits pairs that are not subject to the blockade effect (vertexes of the UDG not paired by an edge) and the unit disk radius r for interacting qubits (adjacent vertexes in the UDG).



(a) Adjacency matrix



(b) Unfeasible graph embedding



(c) Feasible graph embedding

Fig. 1. Representation of unit disk graphs in a quantum register. Whenever two circles intersect an edge is generated. The radius of the circles in figs.1b,1c is the half blockade radius, *i.e.* r. The adjacency matrix in fig.1a describes the desired edges, it determines the feasibility of the embedding.

II. Related work

Since the embedding on the neutral atoms quantum architecture corresponds to finding CUDG solutions and our methodology relies on neural networks, we investigated literature under two main topics. On one hand, state-of-the-art approaches and results in the context of UDG problem solutions are analyzed by pointing out their limitations concerning our use case. On the other hand, previous work targeted the solution of optimization problems through NNs, so they provide useful insights when attempting similar approaches.

A. Solving the unit disk graph problem

Solving the CUDG problem is a complex task on different levels. Indeed solving the UDG recognition problem, *i.e.* determining if a given graph has a UDG realization, is NP-hard [20]. Moreover, even retrieving an approximate solution to the UDG problem is impossible in polynomial time unless $P = NP$ [21]. The UDG problems do not become easier to solve for simplified graphs subclasses, such as for outerplanar [22] and tree graphs [23].

Despite the complexity of the problem, previous works proposed approximation algorithms, especially in the research field of wireless communication, computing virtual coordinates of sensor networks [24]–[27]. Unfortunately, for the quantum application targeted by this paper, the proposed approximations are not feasible. Furthermore, the additional requirements deriving from the quantum device introduced more non-convex constraints. Thus, the standard approach for the CUDG problem solution requires defining the non-convex programming model and trying to solve it with classical solvers. On this side, different formulations for the CUDG problem could be designed, as in [15], and according to the class of the programming model, suitable solvers could be exploited. Mixed-Integer Quadratic Constrained Programming can be solved with *Gurobi* [28] (which nevertheless performed poorly for the formulation proposed in Sec. III-A), and Nonlinear Programming can be approached with *Ipopt* [29]. *Ipopt* solver, due to preliminary better results, has been chosen as a classical solver for comparisons.

B. Applying neural networks to solve optimization problems

Previous studies investigated the application of neural networks to solve optimization problems. In [30], binary Mixed Integer Linear Programs are handled through NNs by devising proper architectures: *ReLu* activation functions implement the binary variables, whilst continuous variables are directly represented by the output value of each unit. The methodology is applied to *feature visualization* and *adversarial machine learning* tasks. A similar approach was presented by Amos and Kolter [31] to solve Quadratic Programs. They provide examples of learning Sudoku problems.

Beyond that, more specific applications of neural networks for optimization are present in the literature. Chandrasekhar and Suresh exploited the weights and biases of NNs to parametrize a density function for topology optimization [32]. In [33], they extended their work to deal with multi-material topologies. In [34], Reinforcement Learning enhances the solution of Capacitated Vehicle Routing Problems, providing a trained policy to solve unseen instances.

Graphs-based optimization (*e.g.*, Vertex Cover, Maximum Independent Set problems) is instead the subject of [35], [36]. The proposed methodologies consider Convolutional and Graph NNs for their optimization purposes.

III. Method

A. The constrained unit disk graph problem

The QC use case, as described in section I, requires a specific formulation of the CUDG optimization problem. Here, we propose a programming model that considers both the unit disk properties and the quantum hardware constraints.

Before diving into a detailed description of the programming model, some remarks are needed. The proposed formulation exploits binary variables to enforce the constraints: a theoretically equivalent model could be designed with only continuous variables. However, binary variables do not affect the convexity of the problem; the formulation with only continuous variables is nonetheless non-convex. Furthermore, these binary variables are needed to deal with the State-of-the-Art solver *Ipopt* [29] through the *Pyomo*[1] Python library. This solver allows constraint violations through tolerance parameter settings, but even the most stringent tolerance does not prevent numerical issues (*e.g.*, numerical cancellation and errors inherent in floating-point arithmetic). Thus it may lead to unfeasible solutions when the feasibility is not enforced through binary variables, explicitly reflecting constraint violations in the objective function. Beware that in this specific quantum application, the requirements correspond to hard constraints, *i.e.* the quantum hardware cannot deal with qubits positions that are approximately feasible.

To mathematically define the constrained unit disk graph (CUDG) problem, we introduce the following notation.
$\mathcal{G}(\mathcal{V}, \mathcal{E})$ represents the undirected graph to embed, with $\mathcal{V}$ as the set of vertexes and $\mathcal{E}$ as the set of undirected edges. The number of vertexes $|\mathcal{V}|$ will be denoted as n, and the vertexes' labels will be indexed starting from 0; $\mathcal{P}$ is the set of all unordered pairs in $\mathcal{V}$, so $|\mathcal{P}| = \frac{n(n-1)}{2}$. The square matrix A of size $n \times n$ is the adjacency matrix of $\mathcal{G}$.
The parameterization of the CUDG problems' instances is determined by the positive constants D_{min}, D_{adj} and L that define the feasibility domain. Respectively, they represent the minimum allowed distance between vertex pairs, the maximum allowed distance between adjacent vertexes, and the maximum radius of the circle/sphere inscribing the graph embedding. Finally, the embedding dimensionality will be defined as $N \in \{2, 3\}$.

Regarding the CUDG programming model, the coordinates of the vertexes are represented by N-dimensional vectors $\overrightarrow{p}_i, \quad \forall i \in \mathcal{V}$. These are continuous variables in the square/cubic domain of side $2L$ (see eq. (2h)). The maximum distance between adjacent pairs is modelled through the continuous variable d_{adj}, eq. (2i). The minimum distance between not adjacent pairs is defined by another continuous variable $d_{\overline{adj}}$, eq. (2j). At last, the binary variables δ_{ij}, $\forall \{i, j\} \in \mathcal{P}$, defined as follows, model the feasibility of the solution.

$$\delta_{ij} := \begin{cases} 1 & \text{pair distance is unfeasible} \\ 0 & \text{pair distance is feasible} \end{cases} \tag{1}$$

It is relevant to notice that the feasibility conditions are modelled accordingly to the adjacency pattern described by A to account for the unit disk graph property and constraints: adjacent vertexes' feasible distances are in the range $[D_{min}, D_{adj}]$ (see Eqs. (2c), (2b)), not adjacent vertexes should have pair distances in the range $[D_{adj} + \epsilon, 2L]$ (see Eqs. (2f), (2e)), with a small value ϵ to avoid a strict inequality formulation that is not allowed by the *Pyomo* library. Constraint (2d), combined with the objective function (2a), enforces the adjacent vertexes to be as close as possible, whereas constraint (2g) enhances the distances between not adjacent vertexes to be the greatest as possible.
The overall CUDG programming model is shown below.

[1] https://pyomo.readthedocs.io

Notice that the constraints on the distances are defined taking into account squared Euclidean distances and that the penalty constant $2L - D_{min} + \iota,\ \iota > 0$, associated with each binary variable in the objective function, favours feasibility over the increasing of the *adjacency gap*, $d_{adj} - d_{\overline{adj}}$.

$$\min_{\overrightarrow{p},\delta,d} \quad (2L - D_{min} + \iota) \sum_{\{i,j\}\in\mathcal{P}} \delta_{ij} + d_{adj} - d_{\overline{adj}} \tag{2a}$$

$$\text{s.t.} \quad ||\overrightarrow{p}_i - \overrightarrow{p}_j||_2^2 \leq D_{adj}^2 + (8L^2 - D_{adj}^2)\delta_{ij} \quad (i,j) \in \mathcal{E}, \tag{2b}$$

$$||\overrightarrow{p}_i - \overrightarrow{p}_j||_2^2 \geq (1 - \delta_{ij})D_{min}^2 \quad (i,j) \in \mathcal{E}, \tag{2c}$$

$$||\overrightarrow{p}_i - \overrightarrow{p}_j||_2^2 \leq d_{adj}^2 \quad (i,j) \in \mathcal{E}, \tag{2d}$$

$$||\overrightarrow{p}_i - \overrightarrow{p}_j||_2^2 \leq 4L^2 + 4L^2\delta_{ij} \quad (i,j) \notin \mathcal{E}, \tag{2e}$$

$$||\overrightarrow{p}_i - \overrightarrow{p}_j||_2^2 \geq (1 - \delta_{ij})(D_{adj} + \epsilon)^2 \quad (i,j) \notin \mathcal{E}, \tag{2f}$$

$$||\overrightarrow{p}_i - \overrightarrow{p}_j||_2^2 \geq d_{\overline{adj}}^2 \quad (i,j) \notin \mathcal{E}, \tag{2g}$$

$$\overrightarrow{p} \in [-L, +L]^{n\times N}, \tag{2h}$$

$$d_{adj} \in [D_{min}, D_{adj}], \tag{2i}$$

$$d_{\overline{adj}} \in [D_{adj} + \epsilon, 2L], \tag{2j}$$

$$\delta \in \{0,1\}^{|\mathcal{P}|} \tag{2k}$$

B. Generating the dataset: basic requirements

The programming model presented in section III-A is solved by comparing two solvers: the *Ipopt* solver and the novel neural-enhanced optimization framework detailed in Sec. III-C. To fairly compare these solvers, a test dataset has been created: it consists of 200 graph instances; in particular, there are 20 samples for each value of $n \in \{10, 20, \ldots, 100\}$.

It is worth mentioning that the generation of the dataset takes into account some necessary conditions regarding the feasibility of the solution to the CUDG problems. The identified necessary conditions follow directly from *Thue's Theorem*, which states that *regular hexagonal packing is the densest circle packing in the plane* [37]. As follows, necessary conditions have been defined for the 2-dimensional case, which corresponds to the most quantum-application-ready setting. Further generalization to the 3-dimensional case or more sophisticated conditions will be the subject of future work.

Concerning the real-world application, the domain parameters values are $D_{min} = 4\ \mu m$ and $D_{adj} \approx 10.26\ \mu m$. So, in the densest packing embedding, the hexagon side is $4\ \mu m$, and adjacent vertexes should lie within a $\approx 10.26\ \mu m$ distance. Thus, to properly embed a complete graph with M vertexes, K_M, all its vertexes should have pair distances $\leq D_{adj}$ (see Fig. 2 on the left), hence property 1.

Property 1 (Maximum clique property): Consider the CUDG problem formulated in section III-A. Let $D_{min} = 4\ \mu m$, $D_{adj} = 10.26\ \mu m$, $L = 50\ \mu m$, $\mathcal{G}(\mathcal{V}, \mathcal{E})$ the undirected graph that defines the CUDG problem and M the number of vertexes in the maximum-sized complete graph (or clique) of $\mathcal{G}$, then a feasible unit disk graph solution can be obtained only if $M \leq 7$.

Moreover, still considering the same regular hexagonal packing combined with the unit disk graph definition, property 2 follows.

Property 2 (Maximum degree property): Consider the CUDG problem formulated in section III-A. Let $D_{min} = 4\ \mu m$, $D_{adj} = 10.26\ \mu m$, $L = 50\ \mu m$, $\mathcal{G}(\mathcal{V}, \mathcal{E})$ the undirected graph that defines the CUDG problem and Δ the maximum degree for vertexes in $\mathcal{G}$, then a feasible unit disk graph solution can be obtained only if $\Delta \leq 18$.

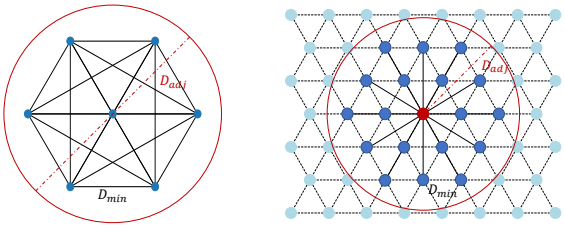


Fig. 2. Representations of a K_7, clique with 7 vertexes, (on the left) and of a graph with maximum degree $\Delta = 18$ (on the right) feasible embeddings, for a CUDG problem with $D_{min} = 4\ \mu m$, $L = 50\ \mu m$ and $D_{adj} = 10.26\ \mu m$.

Fig. 2, on the right, shows a feasible embedding for a subgraph with $\Delta = 18$. All the vertexes lying within the circular red area, with radius $D_{adj} = 10.26\ \mu m$, are neighbours of the red-colored vertex, which among the highlighted vertexes is the one with the highest degree $\Delta = 18$. However, the represented embedding constraints the connectivity of neighbouring vertexes. Thus it is not a feasible embedding for all arbitrary graphs with $\Delta = 18$.

So, since properties 1 and 2 define the necessary conditions for the CUDG to have a feasible solution, they govern the dataset generation. To have more chance and get feasible embeddings, the graphs instances are created by randomly setting initial coordinates $\overrightarrow{c}_i, \forall i \in \mathcal{V}$ in a square domain with side l. Then, edges are defined following a unit disk graph approach, considering a threshold distance d: all vertexes that fall within distance d are paired by edges. No constraints on minimum feasible distance between vertexes or tight relationship between values d and l are considered. So the CUDG problem, as defined in section III-A, is not trivially solved by scaling the initial domain. The parameters l and d increment along with the number of vertexes. The dataset creation is performed iteratively until all the desired samples are obtained. In the specific case, the overall dataset, accounting for 200 graphs, was computed in $\approx 2.30\ min$.

Therefore, in generating the dataset, all the graphs' instances were required to satisfy the following conditions:

- size of the maximum estimated[2] clique ≤ 7;
- maximum degree ≤ 18;
- all vertexes of the graph belong to the same connected component.

C. Neural-enhanced optimization framework

As classical solvers' performance significantly decreases with CUDG problem dimensionality, we designed a novel

[2]Finding the maximum clique is an NP-hard problem, which solution was not targeted in this context, the *NetworkX* maximum clique approximation algorithm was exploited at this scope [38].

methodology to enable solving more CUDG instances. This approach exploits neural networks' capability to approximate non-linear functions to make a modified autoencoder learn the spatial transformation that maps an initial unfeasible solution of the CUDG into a feasible one, possibly satisfying both the unit disk graph property and maximizing the *adjacency gap*. The overall model, named *Distance Encoder Network (DEN)*, takes as input an initial guess on the coordinates $\overrightarrow{p}_i$, learns new feasible coordinates $\overrightarrow{p}_i^f$ in a specific hidden layer, *i.e.* the *coordinates' layer*, and produces as output the squared pair distances $||\overrightarrow{p_i}^f - \overrightarrow{p_j}^f||_2^2,\ \forall\{i,j\} \in \mathcal{P}$. Then, the CUDG problems' constraints and the objective function are handled through a custom loss function, named *Embedding Loss Function (ELF)*.

So, the overall neural-enhanced optimization framework manages one graph instance $\mathcal{G}(\mathcal{V},\mathcal{E})$ at a time. As represented in Figure 3, it consists of two main parts: the **preprocessing phase** provides $\overrightarrow{p}_i,\ \forall i \in \mathcal{V}$, and then the **learning phase** performs the actual optimization to retrieve $\overrightarrow{p}_i^f,\ \forall i \in \mathcal{V}$ for a feasible embedding configuration.

1) The preprocessing phase: the vertex coordinates initialization: Since the proposed optimization framework relies on learning spatial transformations, initial coordinates $\overrightarrow{p}_i, \forall i \in \mathcal{V}$ should be provided. Hence, a preprocessing phase performs the computation of these initial positions. This preliminary phase aims to support the convergence of the optimization algorithm, *i.e.* providing $\overrightarrow{p}_i,\ \forall i \in \mathcal{V}$ that roughly satisfy some of the constraints. Thus, the *DEN* model would converge in fewer iterations (epochs) than when starting from random initialization. Two approaches for $\overrightarrow{p}_i,\ \forall i \in \mathcal{V}$ initialization have been investigated.

- *Scaling method*: all the graphs in the dataset come along with vertex coordinates $\overrightarrow{c}_i,\ \forall i \in \mathcal{V}$. These coordinates can be scaled to the domain of interest to respect at least one set of distance constraints. The choice was made to scale $\overrightarrow{c}_i, \forall i \in \mathcal{V}$ to a circle with radius L, thus retrieving $\overrightarrow{p}_i,\ \forall i \in \mathcal{V}$ initial positions that automatically satisfy constraints (2e). If the target is a 3-dimensional embedding, *i.e.* $N = 3$, then all z-coordinates are initialized to 0.
- *Fruchterman-Reingold method*: this force-directed layout algorithm [39] does not require initial coordinates to be performed, and it models attractive and repulsive forces between vertex pairs according to the adjacency pattern. The chosen Fruchterman-Reingold algorithm implementation is available in the *NetworkX*[3] Python library, where repulsive forces intervene on all vertex pairs with module $k^2/||\overrightarrow{p}_i - \overrightarrow{p}_j||_2^2$, whilst attractive forces intervene only on adjacent pairs and have module $||\overrightarrow{p}_i - \overrightarrow{p}_j||_2/k$. Here the parameter k determines the equilibrium distance at which the two forces balance for adjacent pairs [40], so in this specific setting, it assumes a value of 7 $\mu m \in [D_{min}, D_{adj}]$. This method does not guarantee some constraints satisfaction through $\overrightarrow{p}_i,\ \forall i \in \mathcal{V}$ initialization; though, it allows to deal with graph instances that do not have any initial coordinates to start from, a valid assumption for most of the graphs in UDG-related applications. It handles both 2D and 3D coordinate vectors. This is an iterative method, which tends to converge to a solution in a short time. Thus, it was restrained to 1000 iterations.

[3]https://networkx.org

2) The learning phase: pursuing the feasible embedding: After the **preprocessing phase**, the core of the optimization framework takes place. It consists of the *DEN* model's training. Even if the training algorithm follows the typical approach to training neural networks (forward step to compute outputs and gradients, and backward step to update the network's weights), it has a different meaning. The *DEN* model is supposed to learn a proper spatial transformation that maps an initial not feasible solution of the CUDG problem into a feasible one, still targeting the maximization of the *adjacency gap*. So, the training of each graph sample is independent of the others. Hence, the network architecture is defined according to the problem dimensionality, *i.e.* n and N, and to the desired adjacency pattern, determined by A. To set up a fair comparison among graph samples, for each instance of the CUDG problem, a maximum number of epochs, *i.e. DEN* model trials to find a solution, is fixed, and it is denoted by E.
The *DEN* model architecture includes dropout layers for regularization purposes [41]. Therefore, at each epoch, a *training step* and a subsequent *inference step* are performed: during the *training step*, the dropout works by randomly and temporarily deleting neurons in the hidden layers, then the *ELF* is computed, and *DEN* weights are updated according to *AdamW* optimizer with learning rate defined by the hyperparameter lr [42], after that, in the *inference step* no dropout takes place, and the embedding configurations is computed without further contributing to weights' update.
In the *ELF* definition, the parameter α represents the *adjacency gap*, so it would not be directly optimized if an outer optimization loop were not performed. In the proposed solution, α is initialized to ϵ, and each time the *DEN* model finds a feasible solution, if the new solution corresponds to an increment in the *adjacency gap*, the parameter α assumes the value of the best *adjacency gap*.
This overall procedure, concerning both the feasible embedding retrieval and the *adjacency gap* maximization, is the **learning phase**.

The Distance Encoder Network architecture: the *DEN* model architecture comes from a modification of a typical autoencoder network. It consists of two parts, which are respectively the *trainable autoencoder* component and the *fixed-weight distance calculator*.
The *trainable autoencoder* has the architecture described in table I: the input layer accounts for all the initial coordinates components, $\overrightarrow{p}_i,\ \forall i \in \mathcal{V}$ and the output layer generates the

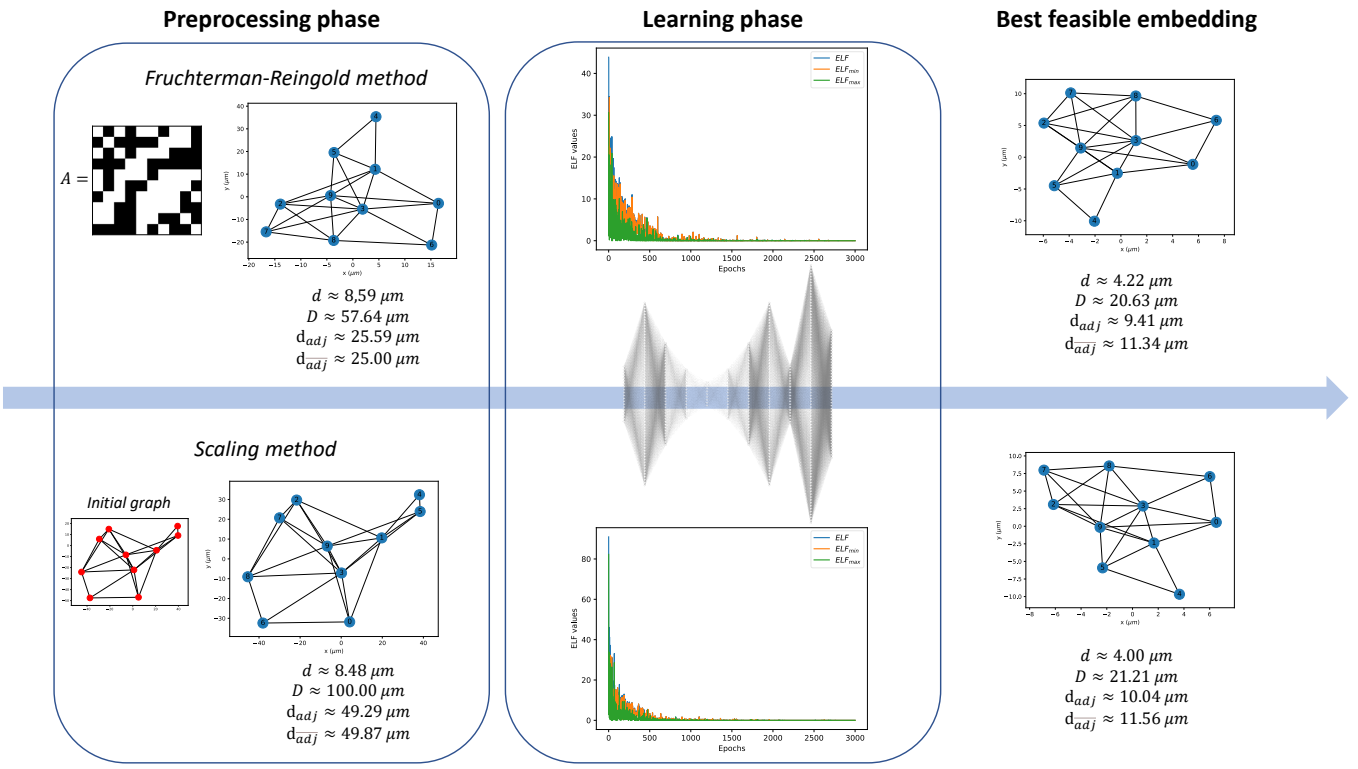


Fig. 3. The optimization framework for the constrained unit disk graph problem: two different approaches are considered for the position initializations in the **preprocessing phase**. Both the *Scaling* and the *Fruchterman-Reingold methods* do not provide initial feasible embeddings. Then, during the **learning phase**, the *DEN* model has 3000 epochs available for learning a proper spatial transformation. Along with the model training, the best feasible embeddings are updated according to the *adjacency gap* maximization goal. The final step returns the best embedding found so far.

transformed coordinates, $\overrightarrow{p_i}^f$, $\forall i \in \mathcal{V}$. In particular, these positions are flattened into 1-dimensional vectors, the input vector I and the output vector O, such that

$$I(k) = \begin{cases} \overrightarrow{p}^{(x)}_k & 0 \leq k < n \\ \overrightarrow{p}^{(y)}_{k-n} & n \leq k < 2n \\ \overrightarrow{p}^{(z)}_{k-2n} & 2n \leq k < 3n \quad \text{and} \quad N = 3 \end{cases} \tag{3}$$

$$O(k) = \begin{cases} \overrightarrow{p^f}^{(x)}_k & 0 \leq k < n \\ \overrightarrow{p^f}^{(y)}_{k-n} & n \leq k < 2n \\ \overrightarrow{p^f}^{(z)}_{k-2n} & 2n \leq k < 3n \quad \text{and} \quad N = 3 \end{cases} \tag{4}$$

In the *trainable autoencoder*, all the fully connected layers allow for the contribution of a bias node and they are equipped with the dropout functionality. This latter is parameterized through the dropout probability hyperparameter p_{drop}.
Finally, the activation function in the last fully connected layer allows generating coordinates in the square (or cubic, when $N = 3$) domain of side $2L$.

The *fixed-weight distance calculator* is the second component of the *DEN* model. It computes squared pair distances, $||\overrightarrow{p_i}^f - \overrightarrow{p_j}^f||_2^2$, $\forall\{i,j\} \in \mathcal{P}$, thus making the *trainable autoencoder* aware that the values contained in O represent Cartesian coordinates whilst providing the proper input to the loss function. To perform this task, the weights of the *distances encoder* are not subject to the training procedure, differently from the *trainable autoencoder*, and the computation of the squared distances is accomplished through the fully connected *difference layer* (input size = $n \times N$, output size = $N \times \binom{n}{2}$, no bias node) followed by a $Square$ activation function and subsequently through the fully connected *sum layer* (input size = $N \times \binom{n}{2}$, output size = $\binom{n}{2}$, no bias node). In particular, the *difference layer*'s weights assume values $\pm 1, 0$, such that the node values u in this layer, before the activation function is applied, are

$$u_{i(n-1)-\binom{i}{2}+j-i-1} = \overrightarrow{p_i^f}^{(x)} - \overrightarrow{p_j^f}^{(x)} \quad \forall\{i,j\} \in \mathcal{P} \tag{5}$$

$$u_{(n-1)(\frac{n}{2}+i)-\binom{i}{2}+j-i-1} = \overrightarrow{p_i^f}^{(y)} - \overrightarrow{p_j^f}^{(y)} \quad \forall\{i,j\} \in \mathcal{P} \tag{6}$$

$$u_{(n-1)(n+i)-\binom{i}{2}+j-i-1} = \overrightarrow{p_i^f}^{(z)} - \overrightarrow{p_j^f}^{(z)} \quad \forall\{i,j\} \in \mathcal{P},\ N = 3 \tag{7}$$

Finally, the weights in the *sum layer* are fixed to values $0, +1$, such that the output values v of the *DEN* models are the squared pair distances in lexicographic order:

$$v_k = \begin{cases} u_k^2 + u^2_{\binom{n}{2}+k} & k \in \left\{0, 1, \ldots, \binom{n}{2} - 1\right\},\ N = 2 \\ u_k^2 + u^2_{\binom{n}{2}+k} + u^2_{2\binom{n}{2}+k} & k \in \left\{0, 1, \ldots, \binom{n}{2} - 1\right\},\ N = 3 \end{cases} \tag{8}$$

TABLE I
Trainable autoencoder COMPONENT ARCHITECTURE FOR A GRAPH WITH n VERTEXES AND TARGETING AN EMBEDDING IN N DIMENSIONS.

	Layer type	Input size	Ouput size	Activation function
Encoder				
	Fully connected layer	$n \times N$	64	$ReLu$
	Fully connected layer	64	36	$ReLu$
	Fully connected layer	36	18	$ReLu$
	Fully connected layer	18	9	$ReLu$
Decoder				
	Fully connected layer	9	18	$ReLu$
	Fully connected layer	18	36	$ReLu$
	Fully connected layer	36	64	$ReLu$
	Fully connected layer	64	$n \times N$	$L \times Tanh$

The Embedding Loss Function: starting from the output of the *DEN* model, $v = ||\overrightarrow{p_i}^f - \overrightarrow{p_j}^f||_2^2, \ \forall\{i,j\} \in \mathcal{P}$, the embedding loss function (*ELF*) is defined to address the feasibility constraints. In particular, it handles separately (2c) (2f) and (2b) (2e). They respectively define lower and upper bounds on feasible distances. So, the $\geq$-based constraints are reflected in the loss ELF_{min}, whilst the $\leq$-based are modelled through the loss ELF_{max}. It is worth mentioning that this modelling approach for inequalities constraints can be exploited beyond the specific CUDG problem.
The ELF_{min} and ELF_{max} definitions exploit the *Margin Ranking loss function*:

$$MarginRanking(v, v^t, m) = avg(max(0, -m(v - v^t))) \quad (9)$$

where m is a vector that determines if we are modelling $\geq$ or $\leq$ inequalities constraints, and the target squared distances v^t are defined according to the adjacency pattern. More precisely, for the ELF_{min} computation: $m = \mathbb{I}_{|\mathcal{P}|}$, and

$$v^t_{i(n-1)-\binom{i}{2}+j-i-1} = \begin{cases} D^2_{min} & A_{i,j} = 1 \\ (D_{adj} + \alpha)^2 & A_{i,j} = 0 \end{cases} \quad \forall\{i,j\} \in \mathcal{P} \quad (10)$$

Whereas, for the ELF_{max}, $m = -\mathbb{I}_{|\mathcal{P}|}$, and

$$v^t_{i(n-1)-\binom{i}{2}+j-i-1} = \begin{cases} D^2_{adj} & A_{i,j} = 1 \\ 4L^2 & A_{i,j} = 0 \end{cases} \quad \forall\{i,j\} \in \mathcal{P} \quad (11)$$

Finally, the overall loss function accounts for both contributions, so

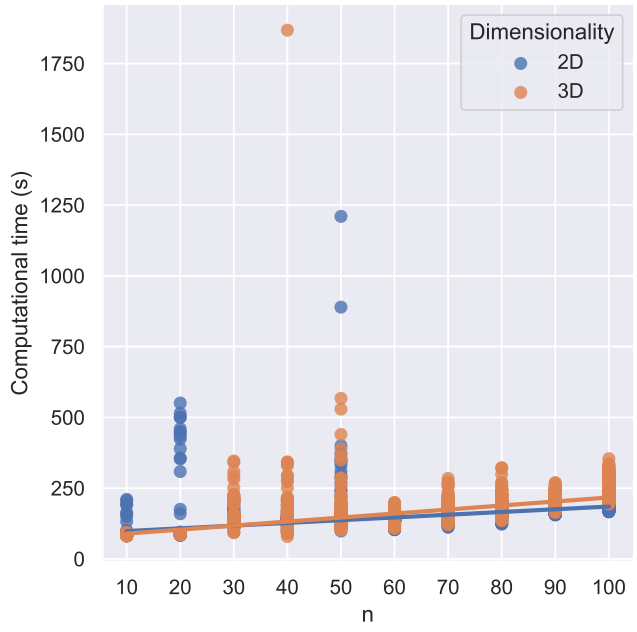
$$ELF(v) = ELF_{min}(v) + ELF_{max}(v) \quad (12)$$

IV. EXPERIMENTS

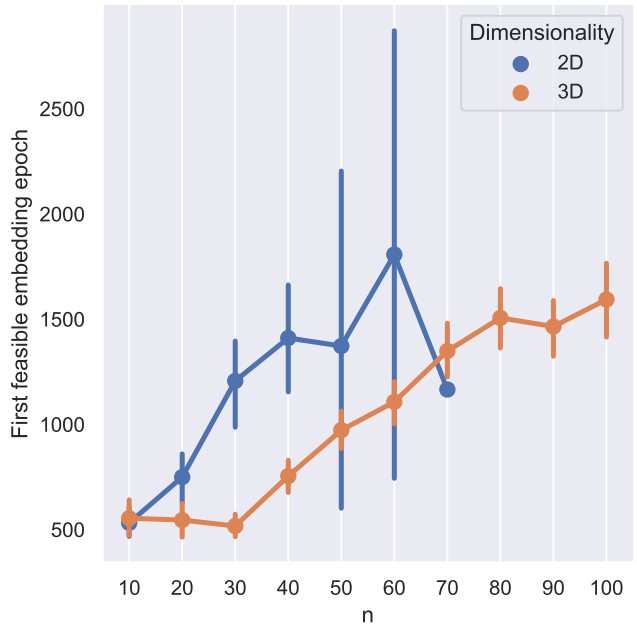
To test the effectiveness of the *DEN*-based optimization framework, 200 graph samples have been considered to define the corresponding CUDG problem instances. The dataset consists of 20 graphs for each value of $n \in \{10, 20, \ldots, 100\}$. The feasible domain has been parametrized according to the characteristics of the quantum hardware of interest, so $D_{min} = 4\ \mu m$, $D_{adj} \approx 10.26\ \mu m$, $L = 50\ \mu m$. The parameters ι and ϵ have been set respectively to 1 and 0.1. Each *DEN* model **learning phase** was allowed to perform $E = 3000$ epochs. The learning rate and the dropout probability hyperparameters were tested with values $lr \in \{0.01, 0.001, 0.0001\}$ and $p_{drop} \in \{0.3, 0.5, 0.7\}$ and combined with the two initialization methods, such as to obtain 18 different trials for retrieving a CUDG solution for each graph in the dataset.
All the trials were run on an IBM Power9-based cluster, with 32 cores/node and 256 GB/node. The nodes in the cluster are also equipped with 4 x NVIDIA Volta V100 GPUs each. However, since the mini-batch size is 1, *i.e.* just one graph sample is considered at each epoch, training on GPUs does not provide a significant acceleration. Whereas the greater availability of CPUs allowed us to exploit better the trials parallelism. In particular, each trial uses 8 physical cores of a node. The overall experiment, comprehensive of the classical solver comparison, required ≈ 5000 core hours.

Figure 4a shows the variability in computational time to perform each one of the $3600 = 200 \times 18$ trials (the dataset consists of 200 graphs and the combinations in the hyperparameters search are 18) for both the embedding dimensionality $N = 2$, $N = 3$, grouped by the number of vertexes n. For each value of n and N, an average computational time, $T_{n,N}$, is retrieved. Then to perform a comparison with the *Ipopt* solver, a multi-start classical optimization takes place, with the number of starting set to 18 and the maximum walltime for each iteration set to $T_{n,N}$, according to the number n of vertexes of the graph instance and the targeted dimensionality N. Figure 4b reports some statics on the first time the *DEN* model find a feasible embedding for each trial, grouping the result by n and separately for dimensionality N. You can notice that, due to the augmented dimensionality, it is easier for the *DEN* model to find solutions in the $3D$ space. On average, the first feasible embedding is found earlier along the epochs. Moreover, as n increases, it becomes more difficult to solve the CUDG problem. As we can see, for $n = 70$, only one graph was successfully embedded within the 3000 epochs in a $2D$ space.

This result is even better represented in Fig. 5 which shows the percentage of feasible embeddings retrieved with each optimization approach. In the case of the *DEN* solver, the results achieved through the two initialization are distinguished. To better prove the advantage of the *DEN* solver, we allowed the *Ipopt* solver to exploit higher computational times. More



(a) Computational time of the **learning phase** at the varying of the $|\mathcal{V}| = n$



(b) Epochs at which the first feasible embedding was found during the **learning phase**

Fig. 4. The average computational times for the *DEN* models training scale linearly with n, and the increment in the dimensionality does not seem to impact significantly on the training duration (Fig. 4a). Whereas the impact of an increased dimensionality, N, is much more evident in Fig. 4b, here the *DEN* model finds more easily a feasible embedding when $N = 3$ than in the case $N = 2$, as can be noticed, feasible embeddings are retrieved up to $n = 100$ and generally the first feasible embedding is found earlier along the epochs.

precisely, we allowed for $10T_{n,N}$ (accordingly to n and N, it could be more than half an hour). Nevertheless, it still did

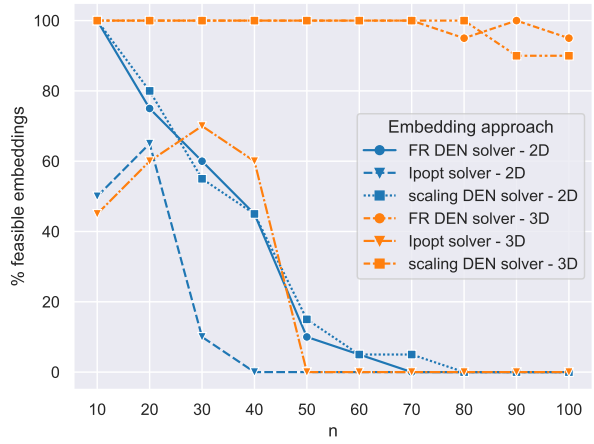


Fig. 5. Percentage of feasible embeddings obtained, on the whole graphs dataset. The *DEN* model outperforms the *Ipopt* solver for both the $N = 2$ and $N = 3$ cases. There is no clear best choice concerning the initialization methods (*Fruchterman-Reingold, i.e. FR*, or *scaling*). As in the case of the first feasible embedding (Fig. 4b), the increment in dimensionality allows for finding more easily feasible embeddings, despite the increment in the models' parameters.

not provide feasible solutions for the instances that were not solved within $T_{n,N}$ walltime.

Fig. 6 illustrates instead the results concerning the maximization of the *adjacency gap*. The *DEN*-based optimization outperforms the classical optimization when the feasibility of the embedding is the goal of major importance. On the other hand, when the *Ipopt*-based solver finds feasible embeddings, they correspond to a greater *adjacency gap*. Possibly, more sophisticated optimization strategies on the α parameter in the *ELF*, combined with an increment of E, could overcome this issue.

A final observation concerns the choice of the hyperparameters. Up to the experiment results, there is no specific combination of value for the learning rate lr and the dropout probability p_{drop} that provides a higher success rate for the CUDG solution. The retrieval of an embedding seems independent of those hyperparameters.

V. Conclusion

This paper presents a novel neural-enhanced optimization framework that addresses a non-convex NP-hard optimization problem, *i.e.*, the constrained unit disk graph problem. It arises from several real-world applications, such as QUBO problems' embedding for neutral-atoms-based quantum hardware.
The proposed *distance encoder network* (*DEN*) model combined with the *embedding loss function* (*ELF*) can find feasible embeddings for a larger set of graphs than the classical solver *Ipopt*. Nonetheless, better embeddings could still be accomplished by improving the *adjacency gap* optimization. Overcoming the limitation concerning *adjacency gap* maximization will be the subject of further work, together with a deeper study on hyper-parameter settings and convergence analysis.

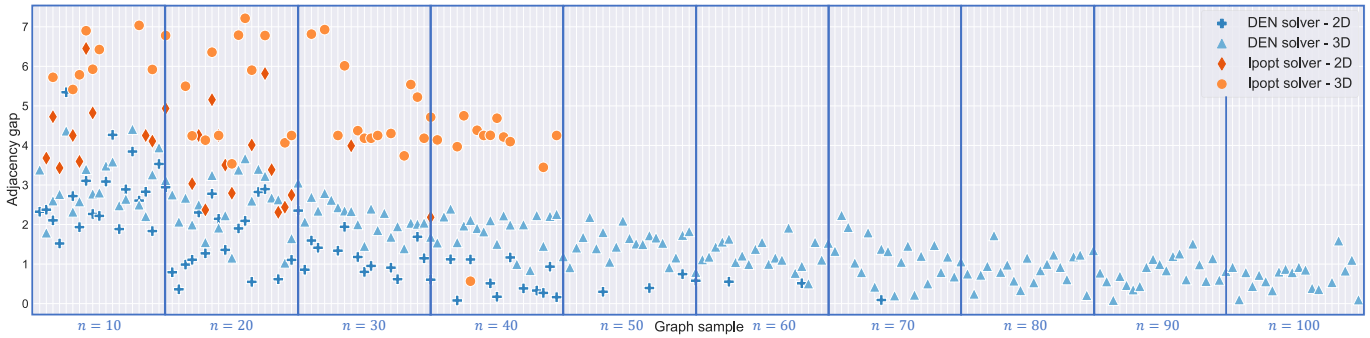


Fig. 6. Comparison of the largest *adjacency gap* values obtained with the different solvers, for all the graph samples in the dataset. The blue rectangles group the graphs' instances by n.

Moreover, modified versions of the *DEN* model and *ELF* will be targeted to provide a more GPU-friendly implementation increasing the mini-batch size during the **training phase**.

A final observation concerns the generality of the approach. The *DEN* model and the *ELF* pursue the computation and optimization of Euclidean distances. Yet, custom modifications can supply outputs of interest for other optimization problems in a similar optimization framework. On this side, we can remark that the definition of the *ELF* function is sufficiently general to model inequality constraints, and the activation functions combined with proper fixed-weight settings allow for representations of objective functions and constraints.

REFERENCES

[1] I. Buluta and F. Nori, "Quantum simulators," *Science*, vol. 326, no. 5949, pp. 108–111, 2009.

[2] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, 2017.

[3] S. McArdle, S. Endo, A. Aspuru-Guzik, S. C. Benjamin, and X. Yuan, "Quantum computational chemistry," *Reviews of Modern Physics*, vol. 92, no. 1, p. 015003, 2020.

[4] G. Kochenberger, J.-K. Hao, F. Glover, M. Lewis, Z. Lü, H. Wang, and Y. Wang, "The unconstrained binary quadratic programming problem: a survey," *Journal of combinatorial optimization*, vol. 28, no. 1, pp. 58–81, 2014.

[5] T. Kadowaki and H. Nishimori, "Quantum annealing in the transverse ising model," *Physical Review E*, vol. 58, no. 5, p. 5355, 1998.

[6] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[7] R. Grimm, M. Weidemüller, and Y. B. Ovchinnikov, "Optical dipole traps for neutral atoms," in *Advances in atomic, molecular, and optical physics*. Elsevier, 2000, vol. 42, pp. 95–170.

[8] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit disk graphs," *Discrete mathematics*, vol. 86, no. 1-3, pp. 165–177, 1990.

[9] M. C. Golumbic, *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

[10] B. Balasundaram and S. Butenko, "Optimization problems in unit-disk graphs." 2009.

[11] Y. Shi, Z. Zhang, Y. Mo, and D.-Z. Du, "Approximation algorithm for minimum weight fault-tolerant virtual backbone in unit disk graphs," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 925–933, 2017.

[12] X.-Y. Li, "Algorithmic, geometric and graphs issues in wireless networks," *Wireless Communications and Mobile Computing*, vol. 3, no. 2, pp. 119–140, 2003.

[13] A. Lucas, "Ising formulations of many NP problems," *Frontiers in Physics*, vol. 2, 2014.

[14] M. F. Serret, B. Marchand, and T. Ayral, "Solving optimization problems with rydberg analog quantum computers: Realistic requirements for quantum advantage using noisy simulation and classical benchmarks," *Physical Review A*, vol. 102, no. 5, p. 052617, 2020.

[15] C. Dalyac, L. Henriet, E. Jeandel, W. Lechner, S. Perdrix, M. Porcheron, and M. Veshchezerova, "Qualifying quantum approaches for hard industrial optimization problems. a case study in the field of smart-charging of electric vehicles," *EPJ Quantum Technology*, vol. 8, no. 1, May 2021. [Online]. Available: http://dx.doi.org/10.1140/epjqt/s40507-021-00100-3

[16] F. Glover, G. Kochenberger, and Y. Du, "A tutorial on formulating and using qubo models," *arXiv preprint arXiv:1811.11538*, 2018.

[17] T. Albash and D. A. Lidar, "Adiabatic quantum computation," *Reviews of Modern Physics*, vol. 90, no. 1, p. 015002, 2018.

[18] D. Ciampini, O. Morsch, and E. Arimondo, "Ultracold rubidium atoms excited to rydberg levels," *Journal of Atomic, Molecular, Condensed Matter and Nano Physics*, vol. 2, no. 3, pp. 161–167, 2015.

[19] C. Picken, R. Legaie, K. McDonnell, and J. Pritchard, "Entanglement of neutral-atom qubits with long ground-rydberg coherence times," *Quantum Science and Technology*, vol. 4, no. 1, p. 015011, 2018.

[20] H. Breu and D. G. Kirkpatrick, "Unit disk graph recognition is NP-hard," *Computational Geometry*, vol. 9, no. 1-2, pp. 3–24, Jan. 1998.

[21] F. Kuhn, T. Moscibroda, and R. Wattenhofer, "Unit disk graph approximation," in *Proceedings of the 2004 Joint Workshop on Foundations of Mobile Computing - DIALM-POMC '04*. Philadelphia, PA, USA: ACM Press, 2004, p. 17.

[22] M. M. Sysło, "Characterizations of outerplanar graphs," *Discrete Mathematics*, vol. 26, no. 1, pp. 47–53, 1979.

[23] S. Bhore, M. Löffler, S. Nickel, and M. Nöllenburg, "Unit Disk Representations of Embedded Trees, Outerplanar and Multi-Legged Graphs," *arXiv:2103.08416 [cs]*, Aug. 2021.

[24] T. Moscibroda, R. O'Dell, M. Wattenhofer, and R. Wattenhofer, "Virtual coordinates for ad hoc and sensor networks," in *Proceedings of the 2004 joint workshop on Foundations of mobile computing*, 2004, pp. 8–16.

[25] S. Pemmaraju and I. Pirwani, "Good quality virtual realization of unit disk graphs," *Journal of Computational Geometry (Old Web Site)*, vol. 2, no. 1, pp. 69–91, 2011.

[26] T. Rusterholz, "On the approximation on unit disk graph coordinates," Tech. rep, Tech. Rep., 2003.

[27] J. Zhou, Y. Chen, B. Leong, and B. Feng, "Practical virtual coordinates for large wireless sensor networks," in *The 18th IEEE International Conference on Network Protocols*. IEEE, 2010, pp. 41–51.

[28] B. Bixby, "The gurobi optimizer," *Transp. Re-search Part B*, vol. 41, no. 2, pp. 159–178, 2007.

[29] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[30] M. Fischetti and J. Jo, "Deep neural networks and mixed integer linear optimization," *Constraints*, vol. 23, no. 3, pp. 296–309, 2018.

[31] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 136–145.

[32] A. Chandrasekhar and K. Suresh, "Tounn: topology optimization using neural networks," *Structural and Multidisciplinary Optimization*, vol. 63, no. 3, pp. 1135–1149, 2021.
[33] ——, "Multi-material topology optimization using neural networks," *Computer-Aided Design*, vol. 136, p. 103017, 2021.
[34] L. Ardon, "Reinforcement learning to solve np-hard problems: an application to the cvrp," *arXiv preprint arXiv:2201.05393*, 2022.
[35] K. Abe, Z. Xu, I. Sato, and M. Sugiyama, "Solving np-hard problems on graphs with extended alphago zero," *arXiv preprint arXiv:1905.11623*, 2019.
[36] E. McCarty, Q. Zhao, A. Sidiropoulos, and Y. Wang, "Nn-baker: A neural-network infused algorithmic framework for optimization problems on geometric intersection graphs," *Advances in Neural Information Processing Systems*, vol. 34, pp. 23 023–23 035, 2021.
[37] H.-C. Chang and L.-C. Wang, "A simple proof of thue's theorem on circle packing," *arXiv preprint arXiv:1009.4322*, 2010.
[38] R. Boppana and M. M. Halldórsson, "Approximating maximum independent sets by excluding subgraphs," *BIT Numerical Mathematics*, vol. 32, no. 2, pp. 180–196, 1992.
[39] T. M. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and experience*, vol. 21, no. 11, pp. 1129–1164, 1991.
[40] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.
[41] S. Wager, S. Wang, and P. S. Liang, "Dropout training as adaptive regularization," *Advances in neural information processing systems*, vol. 26, 2013.
[42] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.