A Hybrid Quantum–inspired Prototype for Solving Nonlinear Continuous Optimization Problems

*

Nimna Suharshani
Department of Mathematics
University of Colombo
Colombo 03, Sri Lanka
2020s18214@ stu.cmb.ac.lk

Awansika Nimuthumana
University of Colombo
School of Computing
Colombo 07, Sri Lanka
awansikanimuthumana@ gmail.com

Geethika Prabhath
University of Colombo
School of Computing
Colombo 07, Sri Lanka
geethikaprabhath1@ gmail.com

Kasun De Zoysa
University of Colombo
School of Computing
Colombo 07, Sri Lanka
kasun@ ucsc.cmb.ac.lk

Anuradha Mahasinghe
Department of Mathematics
University of Colombo
Colombo 03, Sri Lanka
anuradhamahasinghe@ maths.cmb.ac.lk

Nalin Ranasinghe
University of Colombo
School of Computing
Colombo 07, Sri Lanka
dnr@ ucsc.cmb.ac.lk

Asanka Sayakkara
University of Colombo
School of Computing
Colombo 07, Sri Lanka
asa@ ucsc.cmb.ac.lk

***Abstract*—Quantum annealing is a recent advancement in the field of optimization metaheuristics. Quantum annealers such as the D–Wave machines are primarily designed for solving *discrete* optimization problems. Solving nonlinear continuous optimization problems is also a very challenging task, in particular when nonconvex functions and multiple local optima are involved; nevertheless, convex problems can be solved efficiently using standard classical methods. It has not been explored in sufficient detail how quantum annealing could help in solving continuous optimization problems. In this work, we develop a prototype quantum–inspired solver for nonlinear continuous optimization problems. The solver is hybrid, as it has two pathways – *classical* and *quantum–inspired*. Given a continuous optimization problem via the web–based graphical interface that enables users to define and edit optimization problems, our solver computes a nonconvexity index for the problem, which determines the solution pathway. Problems with low nonconvexity indices are sent through the classical pathway and solved by standard separable programming techniques, while the others are transformed into mixed integer linear programming (MILP) problems via separable reformulations, which are restated as quadratic unconstrained binary optimization (QUBO) problems — the problem format compatible for quantum annealers, and eventually sent through the quantum pathway using D–Wave's Neal sampler.**

***Index Terms*—Quantum annealing, Convex functions, Linear programming, Optimization methods, Simulated annealing**

I. Introduction

Continuous optimization problems are fundamental in many domains, including engineering, with applications ranging from finance to machine learning. These problems involve optimizing an objective function subject to constraints, where the decision variable can take any value within a continuous range. This feature distinguishes continuous optimization from discrete or combinatorial optimization, in which variables may be binary, integer, or more abstract objects drawn from sets with finitely many elements [1]. Among continuous problems, nonlinear problems are challenging due to their complexity and computational demands, and many algorithms for nonlinear optimization problems find only a local solution. For convex programming problems, local solutions are also global solutions, avoiding getting trapped in a local optimum. Therefore, it is the nonconvex and nonlinear continuous optimization problems that require highly nonstandard solution techniques.

In recent years, quantum computing has emerged as a new paradigm for solving complex optimization problems. However, its application has been predominantly focused on discrete and combinatorial domains, leaving continuous optimization comparatively underexplored. The main types of quantum approaches to optimization are gate-based variational quantum algorithms (VQAs) and quantum annealing. Gate-based VQAs, such as the Variational Quantum Eigensolver (VQE) and the Quantum Approximate Optimization Algorithm (QAOA), use a classical optimizer to train a parameterized quantum circuit and have shown promise for combinatorial optimization. Nonetheless, they face challenges, including barren plateaus and scalability constraints [2]. Extensions like Continuous-Variable QAOA (CV-QAOA) have adapted this framework for continuous domains, which performs gradient descent in superposition by simulating quantum particle dynamics [3].

In contrast to gate-based VQAs, one can never ignore the success of quantum annealers, particularly D–Wave machines. Unlike the gate–based circuit model of quantum computing, D–Wave machines are particularly designed to solve combinatorial optimization problems and are based on an adiabatic

979-8-3315-6123-9/25/$31.00 © 2025 IEEE

approach. The problem format accessible to a D–Wave machine is known as quadratic unconstrained binary optimization (QUBO) — that is, a Boolean quadratic form which will be minimized using hardware metaheuristics that act as the quantum counterpart of simulated annealing. Though a D–Wave machine solves the QUBO problem, many combinatorial optimization problems can be converted into QUBO format [4]–[11], and promising solutions have been found for many problem instances.

Despite the significant progress achieved in the field of discrete optimization with quantum metaheuristics, it is still not clear how the annealers can be utilized in the field of continuous optimization. We propose to base on a standard nonlinear programming technique, namely, *separable programming*, for this task. Separable programming allows the approximation of nonlinear functions by piecewise linear functions, thus converting the nonlinear problem into a format close to a linear program. When the objective function and the constraints satisfy a set of convexity conditions, the problem can be readily transformed into a linear program, guaranteeing its efficient solvability. However, *nonconvex* optimization problems cannot be restated as linear programs straightaway, yet it is possible to convert them into a mixed integer linear program (MILP) that encodes the solution. The computational cost and the efficiency of this method has been explored in detail in [12]. In this work, we use this MILP reformulation and use it to make the relevant QUBO model, aiming to solve the QUBO problem using a D–Wave machine.

Therefore, we develop a solver with two pathways — the standard separable programming (classical) pathway and the QUBO (quantum) pathway. Given an input optimization problem, our solver checks the convexity of the objective function and the constraints. For practical purposes, using the results in [12], we allow problems with a certain degree of nonconvexity to go through the standard separable programming pathway. In order to do this, we use the nonconvexity index that has been introduced in [12], based on the convexity measures introduced by Davydov et al. [13]. Accordingly, problems with high nonconvexity indices are first converted into MILP, and then into QUBO. We use the PyQUBO library that is designed for constructing QUBO models. In order to find the optimal solution to the QUBO models, we use the Neal package – D–Wave's quantum–inspired solver.

II. Methodology

A. Nonconvexity Index

To determine whether a given optimization problem should be solved via a convex or nonconvex path, a quantitative measure of convexity is required. This study uses the nonconvexity index introduced by Mahasinghe et al. [12] based on the convexity measure by Davydov et al. [13]. For twice continuously differentiable function $h(x)$ on an interval $[a, b]$:

$$C(h, [a, b]) = \frac{\int_a^b \lambda^+(x)\, dx}{\int_a^b |\lambda(x)|\, dx} \tag{1}$$

where $\lambda^+(x) = \max\{h''(x), 0\}$, $\lambda^-(x) = \max\{-h''(x), 0\}$, and $|\lambda(x)| = \lambda^+(x) + \lambda^-(x)$ [12]. This index lies on $[0, 1]$, with 1 indicating full convexity and 0 indicating nonconvexity.

The corresponding nonconvexity index is then defined as:

$$D(h, [a, b]) = 1 - C(h, [a, b]) \tag{2}$$

Given an input problem, our solver computes this index for the objective function and all constraints and guides the selection between a convex optimization path or a nonconvex path.

B. MILP Formulation

A nonlinear optimization problem is separable if both the objective and constraint functions can be written as the sum of a single variable function, as described in [12].

$$\min f(x) = \sum_{j=1}^{n} f_j(x_j) \tag{3}$$

$$g_i(x) = \sum_{j=1}^{n} g_{ij}(x_j) \leq b_i \tag{4}$$

Separable programming approximates these nonlinear functions using piecewise linear functions over grid points within a specified interval $[a_j, b_j]$. For a convex function $f_j(x_j)$, this approximation ensures that any point in the interval can be expressed as a convex combination of two adjacent grid points.

$$x_j = \sum_{l=0}^{k_j} \lambda_{lj}\, x_{lj} \tag{5}$$

$$f_j(x_j) \approx \sum_{l=0}^{k_j} \lambda_{lj}\, f_j(x_{lj}) \tag{6}$$

subject to,

$$\sum_{l=0}^{k_j} \lambda_{lj} = 1, \quad \lambda_{lj} \geq 0 \tag{7}$$

The problem, then reduce to a linear program, can be solved using the simplex method [14], [15]. For variables with linear terms, no approximation is needed.

When the convexity conditions are met, specifically that $f_j(x_j)$ are strictly convex and $g_{ij}(x_j)$ are convex, the adjacency criterion is naturally satisfied [16]–[18]. Thus, the approximated problem can be efficiently solved without additional constraints.

However, when the objective function and constraints deviate from the convexity conditions, a mixed integer linear program can be used. A nonlinear nonconvex separable problem can be approximated by MILP reformulation, the adjacency criterion is replaced by a set of constraints with Boolean variables [19]. Then the approximated problem is as follows:

$$\min f(x) = \sum_{j \in L} f_j(x_j) + \sum_{j \notin L} \sum_{l=0}^{k_j} \lambda_{lj}\, f_j(x_{lj}) \tag{8}$$

subject to,

$$g_i(x) = \sum_{j \in L} g_{ij}(x_j) + \sum_{j \notin L} \sum_{l=0}^{k_j} \lambda_{lj}\, g_{ij}(x_{lj}) \le b_i, \quad i = 1, 2, \ldots, m \tag{9}$$

$$0 \le \lambda_{0j} \le \delta_{0j}, \quad j \notin L \tag{10}$$

$$0 \le \lambda_{lj} \le \delta_{l-1,j} + \delta_{lj}, \quad l = 1, \ldots, k_j; \quad j \notin L$$

$$\sum_{l=0}^{k_j - 1} \delta_{lj} = 1, \quad j \notin L$$

$$\sum_{l=0}^{k_j} \lambda_{lj} = 1, \quad j \notin L$$

$$x_j = \sum_{l=0}^{k_j} \lambda_{lj}\, x_{lj}, \quad j \notin L$$

$$\delta_{lj} \in \{0, 1\}, \quad l = 0, 1, \ldots, k_j - 1, \quad j \notin L$$

$$\lambda_{lj} \ge 0, \quad l = 0, 1, \ldots, k_j, \quad j \notin L$$

$$x_j \ge 0, \quad j \in L$$

1) Example: Consider the nonlinear nonconvex separable programming problem:

$$\begin{aligned} \text{Minimize} \quad & 6x_1^5 + x_1^4 + 2x_1^3 + x_2^2 + x_3 \\ \text{subject to} \quad & x_1 + x_2 \le 6, \\ & x_1 + x_3 \le 3, \\ & x_j \ge 0, \quad j = 1, 2, 3. \end{aligned}$$

The variable x_3 is linear in the problem and therefore does not require a piecewise linear approximation. We decide upper bound for the nonlinear variables without violating any constraints. Therefore, $x_1 \in [0, 3]$ and $x_2 \in [0, 6]$. For the discretization, we use the grid points $[0, 1, 2, 3]$ for x_1 and $[0, 2, 4, 6]$ for x_2:

$$\begin{aligned} x_{01} &= 0, \quad & x_{11} &= 1, \quad & x_{21} &= 2, \quad & x_{31} &= 3, \\ x_{02} &= 0, \quad & x_{12} &= 2, \quad & x_{22} &= 4, \quad & x_{32} &= 6. \end{aligned}$$

The accuracy of the approximation can be improved by increasing the number of grid points.

Original variables in terms of grid points and grid variables:

$$\begin{aligned} x_1 &= 0\lambda_{01} + 1\lambda_{11} + 2\lambda_{21} + 3\lambda_{31}, \\ x_2 &= 0\lambda_{02} + 2\lambda_{12} + 4\lambda_{22} + 6\lambda_{32}. \end{aligned}$$

Then the approximation problem:

$$\begin{aligned} \text{Min} \quad & 0\lambda_{01} + 9\lambda_{11} + 224\lambda_{21} + 1593\lambda_{31} + 0\lambda_{02} + 4\lambda_{12} \\ & + 16\lambda_{22} + 36\lambda_{32} + x_3 \end{aligned}$$

subject to:

$$\begin{aligned} & 0\lambda_{01} + 1\lambda_{11} + 2\lambda_{21} + 3\lambda_{31} + 0\lambda_{02} + 2\lambda_{12} + \\ & \qquad 4\lambda_{22} + 6\lambda_{32} \le 6, \\ & 0\lambda_{01} + 1\lambda_{11} + 2\lambda_{21} + 3\lambda_{31} + x_3 \le 3, \\ & \lambda_{01} + \lambda_{11} + \lambda_{21} + \lambda_{31} = 1, \\ & \lambda_{02} + \lambda_{12} + \lambda_{22} + \lambda_{32} = 1, \\ & 0 \le \lambda_{01} \le \delta_{01}, \\ & 0 \le \lambda_{11} \le \delta_{01} + \delta_{11}, \\ & 0 \le \lambda_{21} \le \delta_{11} + \delta_{21}, \\ & 0 \le \lambda_{31} \le \delta_{21}, \\ & 0 \le \lambda_{02} \le \delta_{02}, \\ & 0 \le \lambda_{12} \le \delta_{12} + \delta_{22}, \\ & 0 \le \lambda_{22} \le \delta_{22}, \\ & 0 \le \lambda_{32} \le \delta_{22}, \\ & \delta_{01} + \delta_{11} + \delta_{21} = 1, \\ & \delta_{02} + \delta_{12} + \delta_{22} = 1, \\ & \delta_{lj} \in \{0, 1\}, \quad l = 0, 1, 2, \quad j = 1, 2, \\ & \lambda_{lj} \ge 0, \quad l = 0, 1, 2, 3, \quad j = 1, 2, \\ & x_3 \ge 0. \end{aligned}$$

C. QUBO Formulation

Quadratic unconstrained binary optimization (QUBO) is a mathematical formulation that can represent a wide range of combinatorial optimization problems [20]. In a QUBO model, all variables are binary, and the objective is to minimize a quadratic function of these variables:

$$\min \quad y = \sum_i Q_{ii} x_i + \sum_{i<j} Q_{ij} x_i x_j \tag{11}$$

where,

- $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ is a vector of binary variables,
- Q is an $n \times n$ matrix of real coefficients,
- Q_{ii} are the the diagonal elements representing the linear coefficients of x_i,
- Q_{ij} are the off–diagonal elements representing interactions between x_i and x_j.

It is common to assume that the matrix Q is either symmetric or in upper triangular form [20]. Symmetric form uses $q_{ij} = (q_{ij} + q_{ji})/2$, while the upper triangular form sets $q_{ij} = q_{ij} + q_{ji}$ for $j > i$ and $q_{ij} = 0$ for $i > j$.

1) Example: Consider the optimization problem:

$$\text{Min} \quad y = 4x_1 - 8x_2 + 5x_3 + 6x_1x_2 + 10x_1x_3 + 4x_2x_3$$

where all variables x_j are binary ($j = 1, 2, 3$).

- $4x_1 - 8x_2 + 5x_3$ is the linear part of this function, and $6x_1x_2 + 10x_1x_3 + 4x_2x_3$ is the quadratic part of this function, which we have to minimize.
- The binary variables satisfy $x_j = x_j^2$, and then the linear part can be written as $4x_1^2 - 8x_2^2 + 5x_3^2$.

- Then we can write the model in the following matrix form:

$$\min \quad y = \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} 4 & 3 & 5 \\ 3 & -8 & 2 \\ 5 & 2 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

- The solution to the above model is $y = -8$, when $x_1 = 0, x_2 = 1$, and $x_3 = 0$.

D. MILP to QUBO Conversion

Mixed integer linear programming (MILP) involves a linear objective function and constraints, with some variables restricted to be integers. In contrast, quadratic unconstrained binary optimization (QUBO) models involve quadratic objectives, binary variables, and there are no constraints. It is suited for both quantum and classical optimization algorithms like quantum annealing. The conversion from MILP to QUBO was motivated by the possibility that, for certain problem sizes, quantum computing hardware such as quantum annealers could solve QUBO problems more efficiently than classical computers.

Consider the above MILP approximated problem in Example II-C1:

First, make all the inequality constraints in the MILP problem to equality constraints by adding slack variables.

$$\begin{aligned} \text{Min} \quad & 0\lambda_{01} + 9\lambda_{11} + 224\lambda_{21} + 1593\lambda_{31} + 0\lambda_{02} + 4\lambda_{12} \\ & + 16\lambda_{22} + 36\lambda_{32} + x_3 \end{aligned}$$

$$\begin{aligned} & 0\lambda_{01} + 1\lambda_{11} + 2\lambda_{21} + 3\lambda_{31} + 0\lambda_{02} + 2\lambda_{12} + \\ & \qquad 4\lambda_{22} + 6\lambda_{32} + s_1 = 6, \\ & 0\lambda_{01} + 1\lambda_{11} + 2\lambda_{21} + 3\lambda_{31} + x_3 + s_2 = 3, \\ & \lambda_{01} + \lambda_{11} + \lambda_{21} + \lambda_{31} = 1, \\ & \lambda_{02} + \lambda_{12} + \lambda_{22} + \lambda_{32} = 1, \\ & \lambda_{01} + s_3 = \delta_{01}, \\ & \lambda_{11} + s_4 = \delta_{01} + \delta_{11}, \\ & \lambda_{21} + s_5 = \delta_{11} + \delta_{21}, \\ & \lambda_{31} + s_6 = \delta_{21}, \\ & \lambda_{02} + s_7 = \delta_{02}, \\ & \lambda_{22} + s_8 = \delta_{12} + \delta_{22}, \\ & \lambda_{32} + s_9 = \delta_{22}, \\ & \lambda_{11} + s_{10} = \delta_{01} + \delta_{11}, \\ & \delta_{01} + \delta_{11} + \delta_{21} = 1, \\ & \delta_{02} + \delta_{12} + \delta_{22} = 1. \end{aligned}$$

In the above approximated problem, the δ variables are already binary variables. The λ variables and slack variables are continuous variables and need to be encoded using binary variables.

$$\text{Continuous variable} = L + \left(\frac{U - L}{2^n}\right) \sum_{i=0}^{n-1} \left(2^{i-1} b_i\right)$$

where L = lower bound, U = upper bound, and we will approximate continuous variables using n binary variables.

All of the variables are now binary after this binary encoded continuous variable was substituted into the approximated problem. The QUBO model is then created by:

$$\text{QUBO} = \text{Objective function + Penalty terms}$$

$$\text{Penalty terms} = P\,(Ax - b)^2,$$

where P is a large positive constant.

$Ax = b$, linear constraints are converted into quadratic penalty terms.

This algorithm was implemented using Python, and the PyQUBO library was used to define variables, the objective function, and constraints to convert the problem into QUBO.

III. Implementation

As the initial step, we developed a computational algorithm that allows a user to input any nonlinear continuous optimization problem to the solver and check nonconvexity index in the objective function and each constraint separately to identify whether the problem is convex or nonconvex. As the second step, we implemented the separable programming technique for the problem and applied the simplex method to get the global optimum solution. The `scipy.optimize.linprog` function was used to apply the simplex method to solve the separable problem. The MILP reformulation technique algorithm was built to solve nonconvex problems using Python. After that, we implemented the code that can convert the MILP problem to a QUBO problem, and using quantum annealing, we can get the solution.

After formulating the QUBO problem, quantum–inspired annealing was used to obtain solutions. Quantum–inspired annealing, implemented on classical hardware, mimics quantum effects like tunneling through stochastic methods, such as quantum Monte Carlo [21]. These algorithms provide heuristic solutions to complex optimization problems and can be run on classical hardware, providing some advantages of quantum annealing. A key example is simulated quantum annealing, which uses classical Monte Carlo techniques to approximate quantum behavior.

A. Quantum–inspired Optimization Platform

D–Wave system is a leading provider of quantum annealing, offering quantum annealing solutions as Software as a Service (SaaS) via the cloud. The D–Wave Ocean SDK provides samplers that work with both classical and quantum hardware and hybrid approaches that combine the best features of both paradigms.

D–Wave system provides cloud–based quantum annealing tools and a variety of samples, including DWaveSampler, DWaveCliquSampler, LeapHybridSampler, LeapHybridCQMSampler, and LeapHybridDQMSampler [22]. These samplers are optimized for different kinds of problems and provide interfaces to model those problems. These consist of the Binary Quadratic Model (BQM), Discrete Quadratic Model

(DQM), and Constrained Quadratic Model (CQM). The BQM model supports methods like QUBO and Ising, and is the most commonly used interface for Binary Quadratic optimization among them.

Neal is a Python library within the D-Wave Ocean SDK, which was used to solve QUBO problems via simulated annealing. Using the `SimulatedAnnealingSample` from Neal and the PyQUBO library, the QUBO matrix was processed on classical hardware to efficiently approximate optimal solutions.

B. Pseudocode for Nonlinear Nonconvex Optimization Path of the Solver

Input:

- Problem type (min/max)
- Objective function $f(\mathbf{x})$
- List of constraints
- Variables: $x_1, x_2, \ldots, x_n$
- Analysis interval: $[a, b]$

Output:

- Binary–encoded QUBO problem
- Solution with original variable values

Problem analysis:
Identify nonlinear variables in the objective and constraints:
for each nonlinear variable x_i do
    Compute second partial derivative $\dfrac{\partial^2 f}{\partial x_i^2}$
    Determine bounds from constraints or defaults
    Generate grid points within bounds
end for
Linear approximation:
for each nonlinear variable x_i do
    Create lambda variables $\lambda_{i,j}$ for each grid point
    Create binary delta variables $\delta_{i,j}$ for adjacency
    Build approximation: $x_i \approx \sum_j \lambda_{i,j}\, g_{i,j}$
end for
Constraint transformation:
for each constraint c do
    Convert to equality form using slack variables s_k
    Substitute lambda approximations
    Add lambda constraints: $\sum_j \lambda_{i,j} = 1,\ \lambda_{i,j} \geq 0\ \forall j$
    Add adjacency constraints for lambda variables (e.g., at most two consecutive $\lambda_{i,j}$ nonzero, linked by $\delta_{i,j} \in \{0, 1\}$)
end for
Binary encoding:
for each variable in the transformed problem do
    If continuous: encode as

$$v \approx \frac{U}{16} \sum_{k=1}^{4} 2^{\,k-1}\, b_k, \qquad b_k \in \{0, 1\}$$

    If lambda: encode as

$$\lambda_{i,j} \approx \frac{1}{16} \sum_{k=1}^{3} 2^{\,k-1}\, b_{\lambda,k}, \qquad b_{\lambda,k} \in \{0, 1\}$$

    Keep $\delta_{i,j}$ as binary variables
end for
QUBO formulation:
Convert objective to QUBO form (substitute encodings)
Convert constraints to penalty terms $P \cdot (\text{violation})^2$
Construct Hamiltonian:

$$H(\mathbf{b}) = \text{objective}(\mathbf{b}) + \sum \text{penalties}(\mathbf{b})$$

Solve with simulated annealing:
Compile QUBO model (matrix $\mathbf{Q}$)
Sample with Neal's simulated annealer to get the best bitstring $\mathbf{b}^*$
Decode best solution
Recover original variable values
Return solution with objective value and variable assignments

C. User Interface

We refactored the original CLI–based solver into a modular, class–oriented architecture to enhance reusability and testing. A web–based graphical interface was developed to eliminate re–entering of complete problem definitions for minor modifications, enabling users to define and edit problems easily, adjust some solver parameters such as the number of binary encoding bits, and visualize results clearly. The application was developed using *Streamlit*, an open–source Python framework. The convexity analysis, convex path, and nonconvex path were integrated into a unified interactive workflow. While preserving the core solver logic, this implementation enhanced the usability of the tool (see Fig. 1).

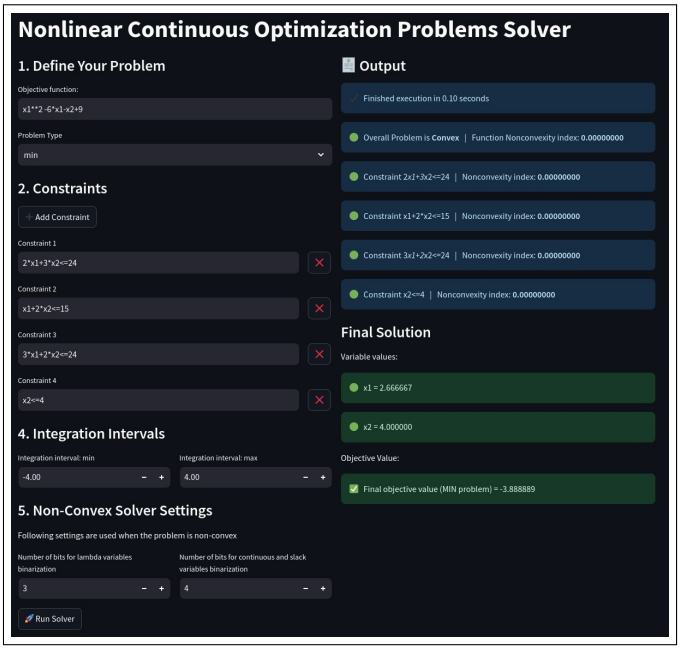

Fig. 1. Interface of the solver.

The application requires a Python environment with the D-Wave Ocean SDK installed. In addition, *Streamlit* must be installed via `pip install streamlit`. Once the dependencies are satisfied, the solver can be launched from the project directory by executing `streamlit run solver_app.py`.

IV. Results and Discussion

Our experimentation of solving a set of nonlinear continuous optimization problems included small–scale test problems, for a few of which the global optimum was known. Problems with total nonconvexity index < 0.1 were directed towards the standard separable programming pathway, while the others were solved using D–Wave Neal. Four grid points based on upper bounds were used for the discretization of each nonlinear term in the separable formulation. As expected, the separable programming pathway resulted in promising solutions. As for the QUBO pathway, each λ variable was binarized by three binary variables, and continuous and slack variables were binarized by four binary variables in the solver. The resulting QUBO was solved using simulated annealing with Neal, using 100 penalty terms and 10,000 reads. As expected, our prototype solver was capable of providing promising solutions to problems with low nonconvexity indices. It is noteworthy that the solver was capable of minimizing the error when providing approximate solutions to problems with high nonconvexity indices. For instance, the optimization problem $\min\ x_1^3 - x_2^2$ subject to $x_1 + x_2 \leq 6,\ x_1, x_2 \geq 0$ has a total nonconvexity index of 1.5, and the exact optimal solution is -36, to which our solver returned the approximate solution -33.05. Further, the problem $\min\ x_1^3 + x_2^2$ subject to $x_1 + x_2 \leq 6,\ 0 \leq x_1, x_2 \leq 4$ has a total nonconvexity index of 0.5 and an exact solution of 0, which was approximated by 0.032470 using our solver. To further validate the performance of the proposed solver, we benchmarked the test problems against Interior Point OPTimizer (IPOPT), a standard nonlinear programming solver, using the NEOS Server. For the above two problems, IPOPT converged to the optimal solutions -36 and 8.35×10^{-10} respectively.

V. Conclusion

This study presented a prototype hybrid quantum–classical solver for nonlinear continuous optimization problems. By developing two distinct pathways—a classical approach for convex problems and a quantum-inspired pathway for nonconvex ones—we demonstrated a practical methodology for leveraging quantum annealing techniques in a continuous domain. The solver integrated separable programming, MILP reformulation, QUBO conversion and quantum annealing to find promising approximate solutions, showcasing the potential of this hybrid framework to approximate global solutions. Although our solver is a prototype which is at the early stage of development, considering the computational hardness of nonlinear optimization problems, we believe the results indicate the potential for a larger–scale implementation of the method we have used. Increasing the number of grid points would make the approximate solutions more accurate. Further, it must be noted that actual quantum annealers are extremely faster and accurate than quantum–inspired solvers like Neal. As no attempt has been made previously to use the MILP formulation based on the separable programming technique, we believe a large–scale implementation of our solver would be helpful in solving industry–scale continuous optimization problems with much accuracy and speed.

References

[1] S. J. Wright., "Continuous optimization (nonlinear and linear programming)," Foundations of Computer-Aided Process Design, Elsevier, New York, 1999.

[2] M. Cerezo et al., "Variational quantum algorithms," Nature Reviews Physics, vol. 3, no. 9, pp. 625–644, Sep. 2021.

[3] G. Verdon, J. M. Arrazola, K. Brádler, and N. Killoran, "A Quantum Approximate Optimization Algorithm for continuous problems," arXiv.org, 2019.

[4] A. Mahasinghe, R. Hua, M. J. Dinneen, and R. Goyal, "Solving the hamiltonian cycle problem using a quantum computer," Proceedings of the Australasian Computer Science Week Multiconference, pp. 1–9, Jan. 2019.

[5] M. J. Dinneen, A. Mahasinghe, and K. Liu, "Finding the chromatic sums of graphs using a D-wave quantum computer," The Journal of Supercomputing, vol. 75, no. 8, pp. 4811–4828, Feb. 2019.

[6] A. Mahasinghe, V. Fernando, and P. Samarawickrama, "QUBO formulations of three NP problems," Journal of Information and Optimization Sciences, vol. 42, no. 7, pp. 1625–1648, Oct. 2021.

[7] F. Glover, G. Kochenberger, and Y. Du, "Quantum Bridge Analytics I: A tutorial on formulating and using Qubo Models," 4OR, vol. 17, no. 4, pp. 335–371, Nov. 2019.

[8] C. Papalitsas, T. Andronikos, K. Giannakis, G. Theocharopoulou, and S. Fanarioti, "A QUBO model for the traveling salesman problem with time windows," Algorithms, vol. 12, no. 11, p. 224, Oct. 2019.

[9] E. Güney, J. Ehrenthal, and T. Hanne, "Qubo formulations and characterization of penalty parameters for the multi-knapsack problem," IEEE Access, vol. 13, pp. 47086–47098, 2025.

[10] P. Codognet, "Comparing QUBO models for quantum annealing: integer encodings for permutation problems," International Transactions in Operational Research, vol. 32, no. 1, pp. 18–37, Apr. 2024.

[11] M. Cattelan and S. Yarkoni, "Modeling routing problems in QUBO with application to ride-hailing," Scientific Reports, vol. 14, no. 1, Aug. 2024.

[12] A. C. Mahasinghe, L. Lakshitha, E. A. Bakare, K. K. W. H. Erandi, "A performance comparison of two nonconvex separable programming techniques," Palestine Journal of Mathematics, vol. 13, no. 4, Oct. 2024.

[13] Y. Davydov, E. Moldavskaya, and R. Zitikis, "Searching for and Quantifying Nonconvexity Regions of Functions*," Lithuanian Mathematical Journal, Nov. 2019.

[14] A. C. Mahasinghe, K. K. W. H. Erandi, and S. S. N. Perera, "An Optimal Lockdown Relaxation Strategy for Minimizing the Economic Effects of COVID-19 Outbreak," International Journal of Mathematics and Mathematical Sciences, vol. 2021, pp. 1–10, Jan. 2021.

[15] A. C. Mahasinghe, K. K. W. H. Erandi, and S. S. N. Perera, "A separable programming approach for resource optimisation in controlling epidemics," International Journal of Operational Research, vol. 48, no. 3, pp. 357–381, 2023.

[16] S. M. Sinha, Mathematical Programming. Elsevier, 2005.

[17] A. C. Mahasinghe, K. K. W. H. Erandi, and S. S. N. Perera, "Optimizing Wiener and Randić Indices of Graphs," Advances in Operations Research, vol. 2020, pp. 1–10, Sep. 2020.

[18] A. C. Mahasinghe, S. S. N. Perera, and K. K. W. H. Erandi, "Optimal Resource Allocation in controlling infectious diseases," Mathematical Methods in Interdisciplinary Sciences, pp. 369–389, Jun. 2020.

[19] M. A. Bolender and D. B. Doman, "Nonlinear Control Allocation Using Piecewise Linear Functions: A Linear Programming Approach," Journal of Guidance, Control, and Dynamics, vol. 28, no. 3, pp. 558–562, May 2005.

[20] F. Glover, G. Kochenberger, and Y. Du, "Quantum Bridge Analytics I: A tutorial on formulating and using Qubo Models," 4OR, vol. 17, no. 4, pp. 335–371, Nov. 2019.

[21] M. Ohzeki, "Quantum Monte Carlo simulation of a particular class of non-stoquastic Hamiltonians in quantum annealing," Scientific Reports, vol. 7, no. 1, Jan. 2017.

[22] "D-Wave Documentation — Python documentation," Dwavesys.com., 2025.