machines

MDPI

Article

HPC: A Computational Benchmark of Classical, Parallel, and Hybrid Metaheuristics for QUBO-Based Suspension Geometry Optimization

Muhammad Waqas Arshad [1,*], Stefano Lodi [1], Omair Ashraf [1], Muhammad Haseeb Rasool [2] and Syed Rizwan Hassan [3,*]

[1] Department of Computer Science and Engineering, University of Bologna, 40126 Bologna, Italy; stefano.lodi@unibo.it (S.L.)
[2] Department of Mechanical Engineering, Purdue University, West Lafayette, IN 47907, USA
[3] Department of Computer Engineering, Gachon University, Seongnam-si 13120, Republic of Korea
* Correspondence: muhammadwaqas.arsha2@unibo.it (M.W.A.); syed9919@gachon.ac.kr (S.R.H.)

Abstract

The calibration of suspension geometry involves highly nonlinear kinematic relationships and leads to challenging optimization landscapes that are difficult to explore efficiently with classical local methods. Quadratic Unconstrained Binary Optimization (QUBO) provides a unified discrete formulation that enables the use of a wide range of metaheuristic solvers, but its practical behavior in realistic engineering-inspired problems remains insufficiently benchmarked. This paper presents a computational benchmarking study of classical, parallel, and hybrid metaheuristic solvers applied to a QUBO-formulated double wishbone suspension geometry problem. A symbolic multi-body kinematic model is constructed and discretized into a large-scale QUBO instance capturing camber and caster tracking objectives across multiple roll conditions. Using a fixed low-resolution binary encoding, we systematically evaluate solver performance in terms of objective value, runtime, and time-to-solution trade-offs. The benchmark includes standard simulated annealing and tabu search, parallel simulated annealing, population-based annealing, bandit-controlled hybrid heuristics, and continuous-relaxation-based ADMM methods with and without annealing refinement. Extensive experiments conducted on a Euro-HPC pre-exascale system demonstrate that parallel and hybrid solvers can achieve substantial reductions in wall-clock time—often exceeding two orders of magnitude—while attaining objective values comparable to classical simulated annealing. The results reveal clear trade-offs between solution quality and computational efficiency, and highlight how solver structure influences performance on large QUBO instances derived from symbolic engineering models. Rather than focusing on final design optimality, this study provides a reproducible reference benchmark and practical insights into solver behavior for QUBO-based engineering optimization problems.

Keywords: Quadratic Unconstrained Binary Optimization (QUBO); metaheuristic optimization; high-performance computing (HPC); solver benchmarking; simulated annealing; hybrid optimization methods; suspension geometry

Academic Editor: Raffaele Di Gregorio

Received: 30 January 2026
Revised: 17 February 2026
Accepted: 20 February 2026
Published: 23 February 2026

1. Introduction

The design and calibration of suspension systems play a central role in vehicle dynamics and ride performance, as they influence handling, comfort, and tire contact conditions

under diverse operating conditions [1]. Traditional suspension tuning often relies on gradient-based or local search methods that assume smooth, convex optimization landscapes, yet real-world kinematic objectives such as camber and caster responses under roll are highly nonlinear and pose significant challenges to classical approaches [2]. At the same time, combinatorial optimization frameworks like Quadratic Unconstrained Binary Optimization (QUBO) have emerged as a flexible representation for discrete approximations of complex engineering problems, enabling the application of a broad spectrum of metaheuristic solvers [3].

QUBO formulations have gained traction in both theoretical and applied optimization research, underpinning advances in quantum-inspired computing [4], large-scale heuristic search [5], and combinatorial engineering design [6]. Numerous studies have explored the use of simulated annealing, tabu search, genetic algorithms, and hybrid methods to tackle high-dimensional binary programs with intricate objective functions [7,8]. Concurrently, the advent of pre-exascale and HPC resources such as the EuroHPC Leonardo system has enabled unprecedented exploration of solver performance on large-scale QUBO instances, highlighting the importance of parallel and distributed strategies [9,10]. Despite these developments, there remains a lack of comprehensive benchmarking studies that systematically compare solver behaviors on QUBO problems grounded in representative engineering models rather than artificial or purely synthetic benchmarks.

In many engineering domains, including suspension geometry, the mapping from continuous design variables to discrete binary representations introduces additional complexity that interacts with solver characteristics in subtle ways [11]. Prior works on QUBO-based engineering optimization have often focused on proof-of-concept formulations [12,13] or on specific solver proposals without contextualizing them within a broader performance landscape [14]. Moreover, evaluations are typically limited to objective values or toy problem sizes, without addressing key practical questions such as runtime efficiency, time-to-solution trade-offs, or the impacts of hybrid strategies in high-performance computing environments. These gaps suggest the need for a systematic and reproducible assessment of how classical, parallel, and hybrid metaheuristics perform on QUBO instances that arise from realistic engineering-inspired models.

To address this need, we present a computational benchmarking study of metaheuristic solvers applied to a QUBO-formulated suspension geometry problem. We construct a symbolic multibody kinematic model of a double wishbone suspension, translate it into a large-scale QUBO instance using a fixed binary discretization, and evaluate a diverse suite of solvers including simulated annealing, tabu search, parallel annealing, population-based methods, bandit-controlled hybrids, and continuous-relaxation-based approaches. Our work does not claim optimal engineering design per se, but instead provides insights into solver efficiency and objective behavior across a representative engineering-derived QUBO, highlighting trade-offs that are often obscured in smaller or synthetic benchmarks.

The contributions of this paper are threefold: (1) a reproducible symbolic-to-QUBO pipeline for large-scale suspension-related objectives; (2) a wide-ranging comparative study of classical, parallel, and hybrid metaheuristic solvers executed on a pre-exascale HPC platform; and (3) an analysis of solver performance in terms of runtime, solution quality, and practical trade-offs that underpin informed solver selection for future QUBO-based engineering applications.

Unlike synthetic or randomly generated QUBO instances that are commonly used in benchmarking studies, the problem considered here arises from a structured symbolic engineering model. As a result, the corresponding QUBO encodes dense constraint couplings, heterogeneous penalty magnitudes, and non-uniform variable sensitivities linked to physical geometry. These characteristics produce solver behaviors that are not typically visible

in small-scale or purely artificial test problems, particularly with respect to penalty dominance, basin structure, and runtime–quality trade-offs. The present study therefore reveals how solver classes interact with realistic engineering-derived QUBO structure rather than idealized combinatorial landscapes. It is important to emphasize that this work is intended as a solver-behavior benchmark rather than a finalized suspension design optimization study. The focus is on comparative algorithmic performance under a fixed formulation and discretization, not on delivering a production-ready geometry. In this context, the explicit low-resolution encoding adopted here serves as a controlled experimental setting that isolates solver differences while maintaining tractable computational complexity.

The remainder of this paper is organized as follows. Section 3 presents the symbolic suspension kinematic model, the continuous camber–caster tracking objective, and the explicit low-resolution discretization adopted in this study. Section 4 details the construction of the Quadratic Unconstrained Binary Optimization (QUBO) formulation, including the binary encoding and penalty structure. Section 5 introduces the solver suite, covering classical, parallel, and hybrid metaheuristic approaches. Section 6 describes the high-performance computing environment, solver configurations, and reproducibility protocol. Section 7 reports and analyzes the benchmarking results, followed by a discussion of key findings and implications in Section 8. Finally, Section 9 summarizes the main conclusions and outlines directions for future work.

2. Related Work

2.1. Suspension Kinematic Synthesis and Hardpoint Optimization

Kinematic synthesis of automotive suspensions has a long history in mechanism design, but recent work increasingly emphasizes optimization-driven hardpoint tuning under tight packaging constraints and performance targets (e.g., camber/caster trends, toe control, roll-center behavior, and compliance proxies). A representative direction is the use of multi-objective formulations that explicitly trade off competing handling objectives while enforcing feasibility through constraints and penalty terms. Authors propose a systematic optimization-based design pipeline for double-wishbone hardpoints, demonstrating that optimization can improve kinematic performance while respecting practical geometric limits [15]. Related efforts also combine kinematic modeling with parametric sensitivity and search strategies to obtain feasible, high-performing hardpoint sets in complex design spaces [16–18].

A recurring theme in the recent literature is that the design landscape is nonconvex, highly coupled, and often discrete in practice due to manufacturing, packaging, and tolerance considerations. This has motivated both (i) robust or constrained formulations that keep solutions physically realizable, and (ii) scalable optimization strategies that can explore large spaces without excessive human tuning [15,19]. In this context, formulating suspension synthesis as a structured optimization problem—rather than manual geometric tuning—has become a widely adopted baseline for modern suspension hardpoint development [20].

2.2. QUBO/Ising Modeling for Engineering Design and Constrained Objectives

Quadratic Unconstrained Binary Optimization (QUBO) and its Ising equivalent have become standard "target forms" for mapping constrained optimization problems into binary decision variables. Their practical appeal comes from (i) a single objective in a unified quadratic form, and (ii) compatibility with a wide set of solvers, including classical metaheuristics, hybrid workflows, and quantum-inspired or quantum-annealing backends. In engineering design, the main modeling challenge is not the quadratic form itself, but the construction of penalty terms that reliably enforce constraints while keeping the energy

landscape navigable [15]. Recent review work highlights that, in real applications [21], solver performance can be dominated by modeling choices—variable encodings, scaling, and penalty calibration—often more than by the choice of optimizer alone [22].

2.3. Annealing-Style Solvers and Neighborhood Search

For large QUBOs, annealing-style heuristics remain a practical baseline because they are simple, robust, and easy to parallelize. Simulated annealing (SA) is widely used as a dependable reference method, while tabu search offers a complementary intensification mechanism through memory-based neighborhood exploration. These methods are widely available in modern BQM/QUBO toolchains and are commonly used for benchmarking and as components in hybrid strategies [23,24]. Hybrid orchestration frameworks combine SA and tabu (and optionally QPU calls) to obtain stronger anytime performance, especially on rugged landscapes where either method alone may stall [25].

A key practical point in contemporary work is that solver speed and objective value are not sufficient: engineering problems typically require evaluating task-level metrics (e.g., camber/caster RMSE against target curves) computed from the decoded continuous design. As a result, many papers emphasize end-to-end pipelines: modeling $\rightarrow$ solving $\rightarrow$ decoding $\rightarrow$ KPI validation, rather than reporting energy alone [15].

2.4. Hybrid and Adaptive Metaheuristics: From Deterministic Hybrids to Learning-Based Selection

Recent research increasingly focuses on hybrid and adaptive combinations of heuristics to improve robustness across problem instances. Deterministic hybrids (e.g., SA followed by tabu intensification) are common, but adaptive schemes go further by selecting operators or solvers based on progress signals (energy improvement, stagnation, acceptance rates, or runtime-normalized gains). Such strategies align well with engineering QUBOs, where different phases of search may benefit from different dynamics: exploration early, intensification late, and occasional restarts to escape local minima [25].

Quantum-inspired annealing has also matured as a pragmatic middle ground between classical heuristics and hardware quantum annealers, targeting QUBO structure while running on classical hardware. Recent work in this direction proposes annealing mechanisms tailored to QUBO that can deliver competitive performance on certain classes of instances [26]. At the same time, review studies caution that strong performance often depends on careful scaling, embedding/encoding choices, and instance characteristics, reinforcing the importance of transparent reproducibility checklists and solver settings [22].

2.5. Software Ecosystems Enabling Reproducible QUBO Experimentation

Finally, the growth of open-source ecosystems has substantially lowered the barrier to building and benchmarking QUBO pipelines. The Ocean toolchain provides mature abstractions for Binary Quadratic Models (BQMs), QUBO compilation, and common samplers (SA, tabu, and hybrid workflows), enabling reproducible comparisons across solvers and hardware settings [23–25,27]. This software support is particularly important for engineering design studies, where end-to-end reproducibility must include not only the solver, but also the encoding, decoding, and evaluation metrics used to validate physical performance.

3. Problem Formulation

In this section we formalize the kinematic design problem that we later map to a QUBO instance. We first introduce a symbolic 3D suspension model, then define the continuous optimization objective in terms of camber and caster targets, and finally state the explicit low–resolution discretization assumption that makes the problem amenable to binary encoding.

Figure 1 illustrates the three-dimensional kinematic structure of the double wishbone suspension considered in this study. The upper control arm (UCA) and lower control arm (LCA) are modeled as rigid triangular linkages connecting inboard chassis hardpoints to the upright, while the steering upright defines the steering axis and wheel-plane orientation. The tie-rod connects the steering rack to the upright and contributes to the overall kinematic constraints. All geometric variables used in the symbolic formulation, including the inboard joint locations $\mathbf{o}_{\mathrm{UB}}$, $\mathbf{o}_{\mathrm{LB}}$, $\mathbf{o}_{\mathrm{SB}}$ and the relative link vectors $\mathbf{p}_{1U}$, $\mathbf{o}_{\mathrm{HL}}$ and $\mathbf{p}_{2S}$, are defined with respect to the vehicle-fixed coordinate frame $\{X, Y, Z\}$ shown in the Figure 1. This schematic representation is consistent with standard double wishbone kinematic models used in suspension analysis and optimization.

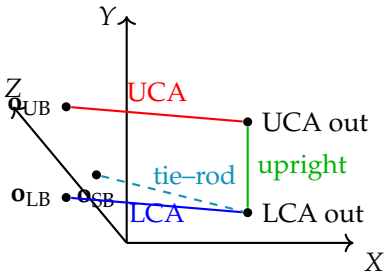

Figure 1. Schematicthree-dimensional kinematic model of the double wishbone suspension considered in this work. The upper control arm (UCA), lower control arm (LCA), steering upright, and tie–rod are shown together with the inboard joint locations $\mathbf{o}_{\mathrm{UB}}$, $\mathbf{o}_{\mathrm{LB}}$, and $\mathbf{o}_{\mathrm{SB}}$. The vehicle-fixed coordinate frame $\{X, Y, Z\}$ is indicated for reference. This diagram illustrates the geometric variables used in the symbolic formulation and subsequent QUBO encoding.

3.1. Symbolic Suspension Model

We consider a front corner of a double–wishbone suspension with a steered upright and a tie–rod. All quantities are expressed in a vehicle–fixed coordinate frame $\mathcal{F} = \{X, Y, Z\}$, where X is longitudinal (forward), Y is lateral (to the left), and Z is vertical (upwards). Additional details can be found in Appendix A.

3.1.1. Geometric Design Variables

Following the notation used in our implementation, we represent the nominal inboard locations and relative link directions by the vectors

$$\begin{aligned}
\mathbf{o}_{\mathrm{UB}} &= (oUB_x, oUB_y, oUB_z)^\top, \\
\mathbf{o}_{\mathrm{LB}} &= (oLB_x, oLB_y, oLB_z)^\top, \\
\mathbf{o}_{\mathrm{SB}} &= (oSB_x, oSB_y, oSB_z)^\top, \\
\mathbf{p}_{1U} &= (p1U_x, p1U_y, p1U_z)^\top, \\
\mathbf{p}_{1H} &= (p1H_x, p1H_y, p1H_z)^\top, \\
\mathbf{p}_{2H} &= (p2H_x, p2H_y, p2H_z)^\top, \\
\mathbf{p}_{2S} &= (p2S_x, p2S_y, p2S_z)^\top, \\
\mathbf{o}_{\mathrm{HL}} &= (oHL_x, oHL_y, oHL_z)^\top, \\
\mathbf{p}_{\mathrm{rear},U} &= (p_\mathrm{rear}, U, x, p_\mathrm{rear}, U, y, p_\mathrm{rear}, U, z)^\top, \\
\mathbf{p}_{\mathrm{rear},L} &= (p_\mathrm{rear}, L, x, p_\mathrm{rear}, L, y, p_\mathrm{rear}, L, z)^\top.
\end{aligned} \tag{1}$$

The three vectors $\mathbf{o}_{\mathrm{UB}}$, $\mathbf{o}_{\mathrm{LB}}$ and $\mathbf{o}_{\mathrm{SB}}$ define the inboard ball–joint locations of the upper control arm (UCA), lower control arm (LCA) and steering rack, respectively. The vectors $\mathbf{p}_{1U}$ and $\mathbf{o}_{\mathrm{HL}}$ point from the inboard pivots to the nominal outboard locations of the UCA and LCA, while $\mathbf{p}_{2S}$ points from the inner steering joint to the outer tie–rod joint. $\mathbf{p}_{1H}$

and $\mathbf{p}_{2H}$ encode the offsets from the upright attachment points to the wheel center, and $\mathbf{p}_{\text{rear},U}$ and $\mathbf{p}_{\text{rear},L}$ define rear inboard mounts used to enforce realistic arm lengths and anti–dive/anti–squat characteristics.

We collect all geometric design variables into a single vector

$$\mathbf{x}_{\text{geom}} = \left[\mathbf{o}_{\text{UB}}^{\top}, \mathbf{o}_{\text{LB}}^{\top}, \mathbf{o}_{\text{SB}}^{\top}, \mathbf{p}_{1U}^{\top}, \mathbf{p}_{1H}^{\top}, \mathbf{p}_{2H}^{\top}, \mathbf{p}_{2S}^{\top}, \mathbf{o}_{\text{HL}}^{\top}, \mathbf{p}_{\text{rear},U}^{\top}, \mathbf{p}_{\text{rear},L}^{\top}\right]^{\top} \in \mathbb{R}^{30}. \tag{2}$$

In the symbolic model we treat each scalar component, such as oUB_x or $p2S_z$, as a distinct continuous variable. The feasible set for $\mathbf{x}_{\text{geom}}$ is restricted to a small box around a hand–designed baseline suspension $\mathbf{x}_{\text{geom}}^{0}$:

$$x_i \in [x_i^0 - \Delta_i,\, x_i^0 + \Delta_i], \qquad i = 1, \ldots, 30, \tag{3}$$

where Δ_i is a geometry–specific investigation range (1 mm in our experiments).

3.1.2. Kinematic State and Input Variables

The wheel motion is parameterized by a low–dimensional vector of generalized coordinates

$$\boldsymbol{\Theta} = \left[\Theta_{\text{U},x}, \Theta_{\text{U},y}, \Theta_{\text{U},z}, \Theta_{\text{L},x}, \Theta_{\text{L},y}, \Theta_{\text{L},z}, \Theta_{\text{TR},x}, \Theta_{\text{TR},y}, \Theta_{\text{TR},z}\right]^{\top}, \tag{4}$$

which represent small rotations of the upper and lower arms and the tie–rod about the X-, Y- and Z-axes. In the implementation these angles are collected into the symbolic vector `all_vars` together with the geometry variables. To keep the problem compact, we constrain each angle to a small range around zero,

$$\Theta_j \in [-\Theta_{\max}, \Theta_{\max}], \qquad \Theta_{\max} = 15^{\circ}, \tag{5}$$

which is consistent with typical operating ranges of a front suspension.

Vehicle–level roll, steer and heave inputs are represented by the parameter vector

$$\boldsymbol{\theta}_L^{(k)} = \left[\theta_{\text{roll}}^{(k)}, \theta_{\text{steer}}^{(k)}, \theta_{\text{heave}}^{(k)}\right]^{\top}, \qquad k = 1, \ldots, N_p, \tag{6}$$

where N_p is the number of sampled operating points over the roll range. In the present work we prescribe a set of roll angles $\{\theta_{\text{roll}}^{(k)}\}_{k=1}^{N_p}$ and keep steer and heave fixed to zero, i.e., $\theta_{\text{steer}}^{(k)} = \theta_{\text{heave}}^{(k)} = 0$. The corresponding symbolic parameters are stored in the *ThetaL_symbolic* array.

3.1.3. Forward Kinematics and Wheel–Plane Orientation

For each operating point k the roll input induces a small rotation of the chassis about the longitudinal axis. Let $R_x(\theta_{\text{roll}}^{(k)})$ denote the corresponding rotation matrix. The outboard ball–joint positions of the upper and lower arms are then given by

$$\begin{aligned} \mathbf{r}_{\text{U}}^{(k)} &= \mathbf{o}_{\text{UB}} + R_x(\theta_{\text{roll}}^{(k)})\, \mathbf{p}_{1U}, \\ \mathbf{r}_{\text{L}}^{(k)} &= \mathbf{o}_{\text{LB}} + R_x(\theta_{\text{roll}}^{(k)})\, \mathbf{o}_{\text{HL}}, \end{aligned} \tag{7}$$

while the outer tie–rod joint is located at

$$\mathbf{r}_{\text{TR}}^{(k)} = \mathbf{o}_{\text{SB}} + R_x(\theta_{\text{roll}}^{(k)})\, \mathbf{p}_{2S}. \tag{8}$$

The steering upright is modeled as a rigid body connecting the upper and lower outboard joints. Its local Z-axis is aligned with the line joining $\mathbf{r}_{\mathrm{L}}^{(k)}$ and $\mathbf{r}_{\mathrm{U}}^{(k)}$,

$$\mathbf{a}_{\mathrm{SA}}^{(k)} = \frac{\mathbf{r}_{\mathrm{U}}^{(k)} - \mathbf{r}_{\mathrm{L}}^{(k)}}{\|\mathbf{r}_{\mathrm{U}}^{(k)} - \mathbf{r}_{\mathrm{L}}^{(k)}\|}, \tag{9}$$

which defines the steering axis. The wheel center is placed at

$$\mathbf{r}_{\mathrm{WC}}^{(k)} = \mathbf{r}_{\mathrm{L}}^{(k)} + \mathbf{p}_{1H}, \tag{10}$$

and the wheel–plane normal is obtained from the upright orientation. For small rotations the camber and caster angles at operating point k can be expressed as

$$\begin{aligned} \gamma^{(k)}(\mathbf{x}_{\mathrm{geom}}, \boldsymbol{\Theta}) &= \arctan 2\Big(a_{\mathrm{SA},y}^{(k)}, a_{\mathrm{SA},z}^{(k)}\Big), \\ \kappa^{(k)}(\mathbf{x}_{\mathrm{geom}}, \boldsymbol{\Theta}) &= \arctan 2\Big(a_{\mathrm{SA},x}^{(k)}, a_{\mathrm{SA},z}^{(k)}\Big), \end{aligned} \tag{11}$$

where $a_{\mathrm{SA},x}^{(k)}$, $a_{\mathrm{SA},y}^{(k)}$ and $a_{\mathrm{SA},z}^{(k)}$ are the components of $\mathbf{a}_{\mathrm{SA}}^{(k)}$ along the X, Y and Z axes.

To obtain a QUBO–compatible objective, the trigonometric functions and ratios in (7)–(11) are expanded symbolically and approximated using low–order Taylor polynomials around the baseline geometry and small–angle state. Using `SymPy`, we derive two multivariate polynomials

$$\begin{aligned} \hat{\gamma}^{(k)}(\mathbf{x}, \boldsymbol{\Theta}) &= f_{\gamma}^{(k)}(\mathbf{x}, \boldsymbol{\Theta}), \\ \hat{\kappa}^{(k)}(\mathbf{x}, \boldsymbol{\Theta}) &= f_{\kappa}^{(k)}(\mathbf{x}, \boldsymbol{\Theta}), \end{aligned} \tag{12}$$

whose coefficients are stored in the symbolic expressions `CamberAngles_Poly` and `CasterAngles_Poly`. Here $\mathbf{x}$ denotes the concatenation of all continuous decision variables (geometry and internal angles). These polynomial surrogates are differentiable, inexpensive to evaluate, and crucially, they remain polynomial after the discretization of Section 3.3, which allows a direct mapping into QUBO form.

3.2. Continuous Kinematic Objective

The design goal is to choose the suspension geometry such that the resulting camber and caster curves track desired targets over a range of body roll angles. Let $\varphi \in [\varphi_{\min}, \varphi_{\max}]$ denote the roll input, with targets $\gamma^{\star}(\varphi)$ and $\kappa^{\star}(\varphi)$ specified by the vehicle dynamics engineer. In our study we consider a linear camber gain and a gently increasing caster with roll.

Using the polynomial surrogates, the continuous optimization problem can be written as

$$\begin{aligned} \min_{\mathbf{x},\boldsymbol{\Theta}} \; & J(\mathbf{x}, \boldsymbol{\Theta}) = J_{\mathrm{track}}(\mathbf{x}, \boldsymbol{\Theta}) + J_{\mathrm{cons}}(\mathbf{x}, \boldsymbol{\Theta}), \\ \text{s.t.} \quad & \mathbf{x}_{\min} \le \mathbf{x} \le \mathbf{x}_{\max}, \\ & \boldsymbol{\Theta}_{\min} \le \boldsymbol{\Theta} \le \boldsymbol{\Theta}_{\max}, \end{aligned} \tag{13}$$

where J_{track} encodes tracking of camber and caster targets and J_{cons} enforces geometric and kinematic constraints.

3.2.1. Tracking Term

The continuous tracking error is defined as

$$J_{\mathrm{track}}(\mathbf{x}, \boldsymbol{\Theta}) = \int_{\varphi_{\min}}^{\varphi_{\max}} \Big[w_{\gamma} \big(\hat{\gamma}(\mathbf{x}, \boldsymbol{\Theta}; \varphi) - \gamma^{\star}(\varphi)\big)^2 + w_{\kappa} \big(\hat{\kappa}(\mathbf{x}, \boldsymbol{\Theta}; \varphi) - \kappa^{\star}(\varphi)\big)^2 \Big] \, d\varphi, \tag{14}$$

where w_γ and w_κ are scalar weights controlling the relative importance of camber and caster matching. In the implementation we expose w_κ as the scalar parameter `W_caster`, while camber error is given unit weight.

3.2.2. Constraint Penalties

The term J_{cons} aggregates soft penalties for various engineering constraints:

- Length constraints: enforce that the upper and lower arms maintain prescribed lengths between front and rear inboard mounts, and between inboard and outboard joints.
- Articulation limits: penalize configurations where ball–joint separations fall below a minimum or exceed a maximum.
- Packaging constraints: keep outboard joints within a feasible region around the wheel center to avoid interference with tire and bodywork.

Each constraint is represented symbolically as either an equality $c_j(\mathbf{x}, \boldsymbol{\Theta}) = 0$ or an inequality $g_\ell(\mathbf{x}, \boldsymbol{\Theta}) \leq 0$. We convert them into differentiable penalties

$$J_{\text{cons}}(\mathbf{x}, \boldsymbol{\Theta}) = P_{\text{eq}} \sum_j c_j(\mathbf{x}, \boldsymbol{\Theta})^2 + P_{\text{ineq}} \sum_\ell \left[\max\{0, g_\ell(\mathbf{x}, \boldsymbol{\Theta})\} \right]^2, \tag{15}$$

where P_{eq} and P_{ineq} are large positive constants. These correspond to the implementation parameters `P_equality` and `P_inequality`. After substitution of the target curves $\gamma^\star(\cdot)$ and $\kappa^\star(\cdot)$ and of the prescribed roll inputs, all terms in (14) and (15) become polynomials in the continuous design variables.

3.3. Explicit Low–Resolution Discretization

Directly solving the continuous problem (13) is difficult because of its highly non–convex polynomial structure and the integral over the roll range. To obtain a QUBO–compatible formulation, we introduce two levels of discretization: (i) discretization of the roll input and (ii) binary encoding of each continuous design variable.

3.3.1. Discretization of the Roll Range

We approximate (14) using a finite set of roll samples $\{\varphi_k\}_{k=1}^{N_p}$ with uniform spacing $\Delta\varphi = (\varphi_{\max} - \varphi_{\min})/(N_p - 1)$. In our experiments we use $N_p = 5$ points over the range $[-5^\circ, 5^\circ]$, matching the implementation choice `np_val=5`. The integral is then replaced by a Riemann sum,

$$J_{\text{track}}(\mathbf{x}, \boldsymbol{\Theta}) \approx \sum_{k=1}^{N_p} \Delta\varphi \left[w_\gamma \left(\hat{\gamma}^{(k)}(\mathbf{x}, \boldsymbol{\Theta}) - \gamma_k^\star \right)^2 + w_\kappa \left(\hat{\kappa}^{(k)}(\mathbf{x}, \boldsymbol{\Theta}) - \kappa_k^\star \right)^2 \right], \tag{16}$$

where $\gamma_k^\star = \gamma^\star(\varphi_k)$ and $\kappa_k^\star = \kappa^\star(\varphi_k)$. The low resolution $N_p = 5$ is an explicit modeling assumption: we assume that camber and caster curves are sufficiently smooth and approximately linear over the considered roll interval, so that a coarse grid still captures the relevant trends. This approximation keeps the symbolic model compact and avoids an explosion in the number of polynomial terms.

3.3.2. Binary Encoding of Design Variables

Each continuous variable x_i in $\mathbf{x}$ is linearly encoded using B binary variables $\{z_{i,0}, \ldots, z_{i,B-1}\}$:

$$x_i = x_{i,\min} + \frac{x_{i,\max} - x_{i,\min}}{2^B - 1} \sum_{b=0}^{B-1} 2^b z_{i,b}, \qquad z_{i,b} \in \{0, 1\}. \tag{17}$$

In the present paper we use $B = 1$ for all variables in order to highlight the algorithmic comparison between classical and hybrid QUBO solvers. The mapping (17) is applied to every scalar component of the geometry vectors and to all internal angles Θ_j. Substituting (17) into (16) and (15) yields a polynomial in the binary variables only, which can be rearranged into the standard QUBO form

$$\min_{\mathbf{z}\in\{0,1\}^N} \mathbf{z}^\top Q\,\mathbf{z} + c, \tag{18}$$

where Q is a symmetric matrix of quadratic coefficients, c is a constant offset and N is the total number of binary variables (787 in the configuration used for our experiments).

For clarity, Figure 2 summarizes the complete methodology that connects the symbolic suspension model, the continuous objective and the QUBO–level optimization studied in the remainder of the paper.

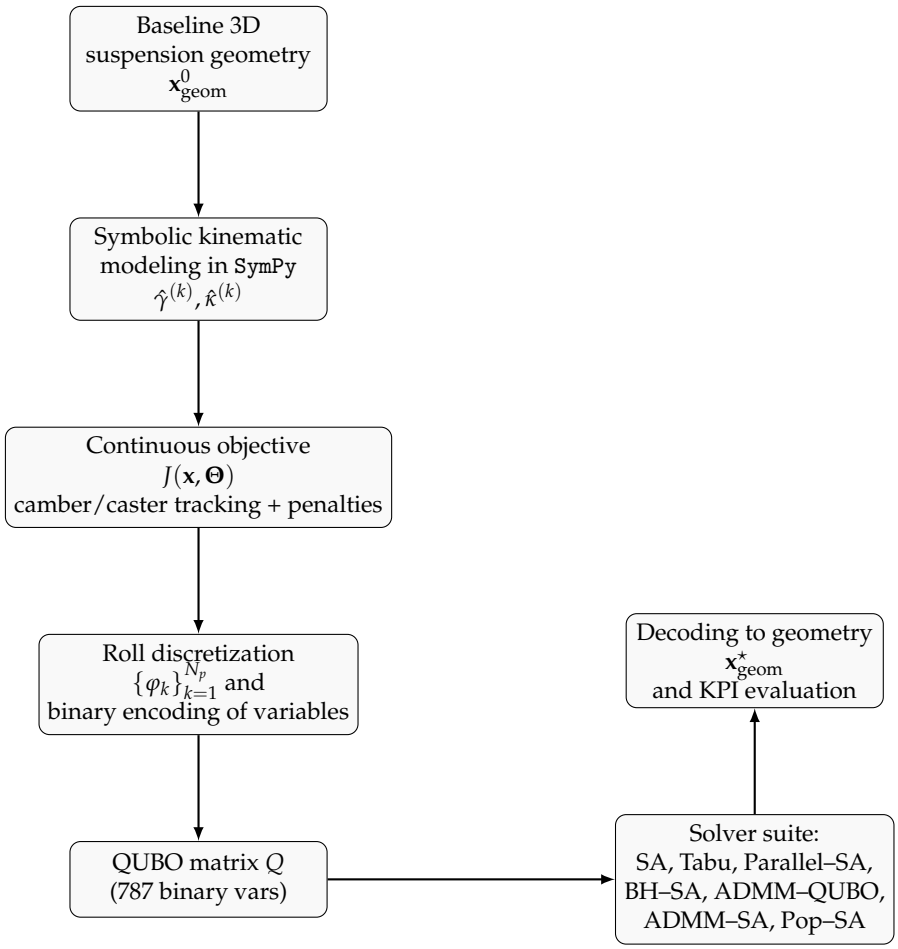

Figure 2. Overview of the proposed methodology. A 3D double–wishbone suspension is encoded as a symbolic polynomial model, from which we derive a continuous camber/caster tracking objective with soft constraints. After discretizing the roll range and linearly encoding each continuous design variable by binary bits, the problem is converted to a QUBO instance. Multiple classical and hybrid QUBO solvers are then compared and the best binary solution is decoded back to a physical suspension geometry.

4. QUBO Construction

This section describes how the continuous, symbolic suspension synthesis problem is transformed into a Quadratic Unconstrained Binary Optimization (QUBO) instance suitable for modern Ising/QUBO solvers. We first introduce the binary encoding used to represent

geometric and state variables, then detail the penalty structure that enforces the kinematic constraints while preserving a purely quadratic binary objective.

4.1. Binary Encoding with Low-Resolution Discretization

Let $\mathbf{v}$ denote the full set of continuous decision variables in the symbolic model, including (i) geometric parameters defining the hardpoint locations and link vectors and (ii) internal state variables such as roll-dependent angles. We write

$$\mathbf{v} \;=\; \big[v_1, \ldots, v_n\big]^\top, \qquad v_i \in [\ell_i, u_i], \tag{19}$$

where the bounds $[\ell_i, u_i]$ are determined from the nominal geometry and a local exploration range (for geometry) or an admissible angular range (for state variables), as implemented in the construction of `var_bounds`.

4.1.1. Binary Representation

Each continuous variable v_i is mapped to a binary string $\mathbf{b}_i \in \{0,1\}^B$ of length $B =$ `NUM_BITS`:

$$\mathbf{b}_i \;=\; \big[b_{i,0}, b_{i,1}, \ldots, b_{i,B-1}\big]^\top, \qquad b_{i,k} \in \{0,1\}. \tag{20}$$

The corresponding integer code is defined as

$$z_i \;=\; \sum_{k=0}^{B-1} 2^k \, b_{i,k}, \qquad z_i \in \{0, 1, \ldots, 2^B - 1\}. \tag{21}$$

The binary-to-continuous decoding is then the standard uniform affine map

$$v_i(\mathbf{b}_i) \;=\; \ell_i \;+\; \frac{u_i - \ell_i}{2^B - 1}\, z_i \;=\; \ell_i \;+\; \frac{u_i - \ell_i}{2^B - 1} \sum_{k=0}^{B-1} 2^k \, b_{i,k}. \tag{22}$$

4.1.2. Specialization to $B = 1$ (NUM_BITS = 1)

In this work we intentionally adopt a low-resolution discretization with

$$B = 1 \quad \Longrightarrow \quad z_i = b_{i,0} \in \{0,1\}, \tag{23}$$

which yields the particularly simple two-level quantization

$$v_i(b_{i,0}) \;=\; \ell_i \;+\; (u_i - \ell_i)\, b_{i,0}. \tag{24}$$

Hence, each variable selects either its lower bound or upper bound. This design choice is justified by three practical considerations aligned with the goal of demonstrating solver-level differences rather than performing a final high-fidelity suspension synthesis:

- Controlling QUBO size: The number of binary variables grows linearly with B and the number of continuous variables. Even at $B = 1$ the resulting instance is already large (hundreds of binary variables after compilation), so increasing B would quickly produce QUBOs that are costly to compile and less suitable for rapid solver benchmarking.
- Benchmarking under strict budgets: The low-resolution encoding creates a challenging combinatorial landscape while keeping runtimes and memory requirements compatible with repeated runs (e.g., for multiple solvers, seeds, and ablations).
- Ablation-friendly baseline: Using $B = 1$ provides a clean baseline for future refinement: the same pipeline directly generalizes to $B > 1$ by replacing (24) with (22).

4.2. Quadratic Form and QUBO Objective

A QUBO is defined over a binary vector $\mathbf{x} \in \{0,1\}^N$ and has the form

$$\min_{\mathbf{x}\in\{0,1\}^N} \; E(\mathbf{x}) \;=\; \mathbf{x}^\top \mathbf{Q}\,\mathbf{x} \;+\; \mathbf{q}^\top \mathbf{x} \;+\; c, \tag{25}$$

where $\mathbf{Q} \in \mathbb{R}^{N\times N}$ is (without loss of generality) upper-triangular, $\mathbf{q} \in \mathbb{R}^N$ is a linear term, and $c \in \mathbb{R}$ is a constant offset. In common software interfaces, (25) is equivalently represented by a dictionary of linear and quadratic coefficients plus an energy offset.

Let $J(\mathbf{v})$ be the continuous objective derived from the symbolic suspension model (e.g., camber/caster tracking loss across roll samples). After substituting inputs and fixed parameters, we obtain a fully symbolic polynomial in the decision variables. Applying the encoding (22) yields a pseudo-Boolean objective

$$\widetilde{J}(\mathbf{x}) \;=\; J\big(\mathbf{v}(\mathbf{x})\big), \tag{26}$$

where $\mathbf{x}$ concatenates all bits from all variables:

$$\mathbf{x} = \big[\mathbf{b}_1^\top, \ldots, \mathbf{b}_n^\top\big]^\top \in \{0,1\}^N, \qquad N = nB. \tag{27}$$

Because the symbolic kinematic expressions typically generate higher-order monomials after expansion, $\widetilde{J}(\mathbf{x})$ may not be quadratic. The compilation step therefore introduces auxiliary variables and quadratic reductions (e.g., Rosenberg-style reductions) to transform the objective into an equivalent quadratic form, resulting in (25). This is the role of the QUBO builder (our `build_qubo` routine), which returns $(\mathbf{Q}, c)$ and an associated decoding model.

4.3. Penalty Structure for Constraint Enforcement

The suspension kinematic model contains constraints that must hold for physical feasibility. These include equality constraints (e.g., rigid link-length preservation, closure constraints) and inequality constraints (e.g., bound constraints or soft feasibility limits). Since QUBO problems are unconstrained, we incorporate constraints via quadratic penalties.

4.3.1. Equality Constraints

Let $\{g_m(\mathbf{v}) = 0\}_{m=1}^M$ denote the set of equality constraints produced by the symbolic model. We enforce them using a sum-of-squares penalty:

$$\mathcal{P}_{\text{eq}}(\mathbf{v}) \;=\; P_{\text{equality}} \sum_{m=1}^{M} g_m(\mathbf{v})^2, \tag{28}$$

where $P_{\text{equality}} > 0$ is a user-defined weight. Squaring converts sign-sensitive violations into nonnegative penalties and is compatible with polynomial expansion and quadratic compilation.

4.3.2. Inequality Constraints

For inequality constraints of the form $h_r(\mathbf{v}) \le 0$, we use a hinge-style penalty:

$$\mathcal{P}_{\text{ineq}}(\mathbf{v}) \;=\; P_{\text{inequality}} \sum_{r=1}^{R} \big[\max(0,\, h_r(\mathbf{v}))\big]^2, \tag{29}$$

with weight $P_{\text{inequality}} > 0$. In practice, the symbolic model introduces slack variables or piecewise polynomial surrogates so that the final penalty remains representable within the QUBO compilation pipeline.

4.3.3. Total Penalized Objective

Combining the tracking objective with penalties yields the constrained-to-unconstrained transformation:

$$J_{\text{tot}}(\mathbf{v}) \;=\; J(\mathbf{v}) \;+\; \mathcal{P}_{\text{eq}}(\mathbf{v}) \;+\; \mathcal{P}_{\text{ineq}}(\mathbf{v}). \tag{30}$$

After substitution of roll inputs and target curves, and applying the binary encoding, we obtain a pseudo-Boolean function

$$E(\mathbf{x}) \;=\; J_{\text{tot}}\big(\mathbf{v}(\mathbf{x})\big), \tag{31}$$

which is then reduced to the quadratic QUBO form (25).

4.3.4. Penalty Selection and Scaling

The penalty weights must be large enough that any constraint violation is energetically dominated by feasible solutions, while not so large as to cause numerical ill-conditioning in solver heuristics. In our implementation, these values are passed as

$$P_{\text{equality}} = \texttt{P\_EQUALITY}, \qquad P_{\text{inequality}} = \texttt{P\_INEQUALITY}, \tag{32}$$

with default settings $P_{\text{equality}} = 1000$ and $P_{\text{inequality}} = 500$ (overridable through environment variables). These values were chosen to (i) strongly discourage infeasible kinematics and (ii) preserve enough dynamic range for the camber/caster tracking objective to meaningfully differentiate feasible designs.

4.4. Resulting QUBO Instance Characteristics

Let n be the number of continuous decision variables and B the number of bits per variable. After discretization, the base number of binary variables is $N = nB$. The quadratic reduction of higher-order monomials introduces additional auxiliary variables, leading to a final QUBO size

$$N_{\text{qubo}} \;\geq\; N, \tag{33}$$

The gap depends on the degree of the polynomial and the way it is reduced. When we set $B = 1$, the compiled QUBO has hundreds of binary variables. This is because of the underlying kinematic complexity and the extra variables needed to make sure the form is strictly quadratic. After that, this QUBO is given to classical samplers (SA, Tabu) and our proposed hybrid solver (Option A) for testing.

5. Optimization Algorithms (Solver Suite)

This section discusses the optimization algorithms that were used to solve the QUBO problems that came from the suspension problem. All solvers use the same Binary Quadratic Model (BQM) that the pipeline in Section 4 makes, which makes sure that the comparison is fair and consistent. The solver suite has classical baselines, parallel versions, and hybrid methods that mix different search strategies that work well together. We don't want to come up with new theoretical solvers; instead, we want to test their real-world performance—runtime, objective quality, and robustness—on a large QUBO instance based on a realistic engineering-inspired model.

5.1. Simulated Annealing (SA)

Simulated Annealing (SA) is a random local-search algorithm that is based on the physical process of annealing. Because it is simple, strong, and widely available in optimization frameworks, it is often used as a baseline solver for QUBO problems.

Given a current binary state $\mathbf{x}$, SA suggests a neighboring state $\mathbf{x}'$ by changing one or more bits. The move is accepted according to the Metropolis criterion

$$\Pr(\mathbf{x} \to \mathbf{x}') = \begin{cases} 1, & \Delta E \leq 0, \\ \exp(-\Delta E/T), & \Delta E > 0, \end{cases} \tag{34}$$

where $\Delta E = E(\mathbf{x}') - E(\mathbf{x})$ and T is a temperature parameter that decreases according to a cooling schedule. At high temperatures, uphill moves are frequently accepted, promoting exploration; as T decreases, the algorithm becomes increasingly greedy.

In our experiments, SA serves as the primary reference method against which all other solvers are compared in terms of both solution quality and runtime.

5.2. Tabu Search

Tabu Search is a deterministic neighborhood-search heuristic that enhances local search by maintaining a short-term memory structure called a tabu list. Recently visited moves or bit flips are temporarily forbidden, which prevents cycling and encourages exploration beyond local minima.

At each iteration, the algorithm evaluates a set of candidate moves and selects the best admissible one, even if it leads to a higher objective value. A tabu restriction may be overridden by an aspiration criterion, for example if the move yields the best solution seen so far.

Tabu Search is often effective at intensifying search around promising regions, but its performance is sensitive to tenure length and neighborhood evaluation cost. We therefore include it primarily as a complementary baseline to SA, rather than as a standalone high-performance method.

5.3. Parallel Simulated Annealing (Parallel-SA)

Simulated annealing is inherently parallelizable because independent runs can be executed with different random seeds. Parallel-SA exploits this property by launching multiple SA instances concurrently and retaining the best solution found.

Let P denote the number of parallel workers. Each worker performs an independent SA run on the same QUBO, and the final solution is selected as

$$\mathbf{x}^\star = \arg \min_{p \in \{1,\ldots,P\}} E(\mathbf{x}_p), \tag{35}$$

where $\mathbf{x}_p$ is the best solution returned by worker p.

Parallel-SA does not change the underlying SA dynamics, but it reduces time-to-solution by increasing the probability that at least one run reaches a low-energy basin. This approach is particularly well suited to HPC environments, where multiple CPU cores are readily available.

5.4. Bandit-Hybrid SA–Tabu (BH-SA)

The Bandit-Hybrid SA–Tabu (BH-SA) solver combines the exploratory strength of SA with the intensification capability of Tabu Search using a simple online decision mechanism. Instead of setting a fixed schedule, BH-SA chooses between SA and Tabu, updates based on how well the system is performing.

The solver is formulated as a two-armed bandit problem, where each arm corresponds to a solver component (SA or Tabu). At each phase, one component is selected and applied

for a short block of iterations. The reward associated with arm a is defined as the energy improvement per unit time,

$$r_a = \max\left(0, \frac{E_{\text{before}} - E_{\text{after}}}{\Delta t}\right). \tag{36}$$

The action values Q_a is updated via an exponential moving average,

$$Q_a \leftarrow (1-\alpha)Q_a + \alpha r_a, \tag{37}$$

with learning rate $\alpha \in (0,1)$. Selection probabilities are obtained via a softmax policy over the current Q values.

This adaptive strategy allows the solver to favour whichever component is currently producing the greatest progress, without requiring manual tuning of a switching schedule.

5.5. Population-Parallel Simulated Annealing (Pop-SA)

Population-Parallel SA (Pop-SA) extends classical SA by maintaining a population of candidate solutions that evolve over time. Each individual undergoes short SA refinements, and information is shared across the population through selection and recombination.

Let $\mathcal{P} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ denote the population. At each generation:

1. Each individual is refined by a short SA run (executed in parallel across available workers).
2. The population is ranked by energy, and a subset of elite solutions is retained.
3. New individuals are generated via uniform crossover between elites, followed by random bit mutations.

This strategy combines the global exploration benefits of population methods with the local refinement capability of SA. Parallel execution across the population makes Pop-SA well suited to HPC platforms.

5.6. ADMM-QUBO

To complement purely combinatorial solvers, we include a continuous relaxation approach based on the Alternating Direction Method of Multipliers (ADMM). The binary constraint $\mathbf{x} \in \{0,1\}^N$ is relaxed to $\mathbf{x} \in [0,1]^N$ and enforced through an auxiliary variable $\mathbf{z} \in \{0,1\}^N$ with the constraint $\mathbf{x} = \mathbf{z}$.

The augmented Lagrangian is

$$\mathcal{L}_\rho(\mathbf{x}, \mathbf{z}, \mathbf{u}) = E(\mathbf{x}) + \frac{\rho}{2}\|\mathbf{x} - \mathbf{z} + \mathbf{u}\|_2^2, \tag{38}$$

where $\mathbf{u}$ is a scaled dual variable and $\rho > 0$ is a penalty parameter. The ADMM iterations alternate between:

$$\mathbf{x}^{k+1} = \arg\min_{\mathbf{x} \in [0,1]^N} \mathcal{L}_\rho(\mathbf{x}, \mathbf{z}^k, \mathbf{u}^k), \tag{39}$$

$$\mathbf{z}^{k+1} = \Pi_{\{0,1\}^N}(\mathbf{x}^{k+1} + \mathbf{u}^k), \tag{40}$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \mathbf{x}^{k+1} - \mathbf{z}^{k+1}, \tag{41}$$

where $\Pi_{\{0,1\}^N}$ denotes component-wise rounding.

ADMM-QUBO converges rapidly to a low-energy binary solution, but the result is generally approximate. Its main strength lies in extremely fast time-to-solution, making it a useful baseline for warm starts and rapid screening.

5.7. ADMM–SA Hybrid

The ADMM–SA hybrid combines the strengths of continuous relaxation and stochastic refinement. First, ADMM-QUBO is used to obtain a low-energy binary solution $\mathbf{x}_{\text{ADMM}}$. This solution is then supplied as the initial state of a short SA run,

$$\mathbf{x}_0 = \mathbf{x}_{\text{ADMM}}, \tag{42}$$

which makes the solution better in the discrete search space.

This hybrid method uses ADMM's speed for coarse optimisation and SA's strength for local improvement. It strikes a good balance between runtime and solution quality.

Table 1 summarizes the conceptual role of each solver in the benchmark.

Table 1. Roles of solvers included in the benchmark.

Solver	Primary Role
SA	Baseline stochastic local search
Tabu	Deterministic intensification baseline
Parallel-SA	Embarrassingly parallel baseline
BH-SA	Adaptive hybrid exploration/intensification
Pop-SA	Population-based parallel exploration
ADMM-QUBO	Fast continuous relaxation
ADMM–SA	Warm-started hybrid refinement

6. Experimental Setup

This section discusses the computer environment, the settings for the solver, and the experimental protocol that were used in the benchmarking study. Reproducibility and transparency are given special attention, in line with the best practices for large-scale computational optimization experiments.

6.1. High-Performance Computing Environment

The Leonardo supercomputing system, which is run by CINECA as part of the EuroHPC Joint Undertaking, was used for all of the experiments. Leonardo is a pre-exascale platform made to handle big scientific and engineering tasks. The CPU partition was used for the experiments described in this paper, which had the following setup with more details in Appendix C:

- System: CINECA Leonardo (EuroHPC pre-exascale system),
- Compute nodes: 32-core AMD EPYC 7H12 processors,
- Memory: 128 GB RAM per node,
- Operating system: Linux (CentOS),

Parallel execution was implemented at the process level using Python's multiprocessing and concurrent.futures interfaces. Each solver instance was assigned dedicated CPU cores, and no GPU or accelerator resources were used. All reported runtimes correspond to wall-clock time measured on exclusive nodes to avoid interference from co-scheduled workloads.

6.2. Solver Configuration and Parameter Settings

To ensure comparability across solvers, all algorithms operated on the same QUBO instance and used consistent stopping criteria wherever possible. The main control parameters include the number of reads, number of sweeps, population size, and iteration counts, depending on the solver type.

For simulated annealing–based methods (SA, Parallel-SA, BH-SA, Pop-SA, and ADMM–SA), the primary parameters are:

- Number of reads: total number of independent samples,
- Number of sweeps: iterations per sample,
- Cooling schedule: default geometric schedule provided by the sampler implementation.

Tabu Search uses a fixed number of reads and an internal tenure mechanism, while ADMM-QUBO relies on a specified number of outer ADMM iterations and inner gradient descent steps. All parameter values were selected to balance runtime and solution quality, and are reported explicitly in the supplementary configuration files for reproducibility. The parameter settings for baseline and parallel solvers can be found in Appendix B.

6.3. Random Seeds and Reproducibility

Metaheuristic solvers are inherently stochastic. To control for randomness and enable exact replication of results, all experiments were conducted with fixed pseudo-random seeds. Specifically, a global seed was set for:

- NumPy random number generation,
- solver-level random initialization,
- parallel worker seed assignment.

By fixing these seeds, repeated executions of the same solver under identical conditions yield identical sequences of random decisions and, consequently, identical solution trajectories. This choice allows direct solver-to-solver comparison without confounding variability due to random fluctuations.

6.4. FAST_MODE Configuration

Two execution modes were implemented in the experimental code: `FAST_MODE` and a full-accuracy mode. In this paper we report results obtained with `FAST_MODE` enabled.

When `FAST_MODE` is active, solver parameters are configured to favor rapid execution and broad comparative coverage rather than asymptotic convergence. Concretely, this mode uses:

- reduced numbers of reads and sweeps for SA-based solvers,
- shorter tabu tenures and timeouts,
- fewer ADMM iterations and gradient descent steps,
- smaller population sizes and fewer generations in Pop-SA.

The rationale for this choice is twofold. First, it reflects practical design scenarios in which time budgets are limited and approximate solutions are acceptable. Second, it enables systematic benchmarking of multiple solvers within a single experimental campaign on a shared HPC platform.

All solvers were tested under the same `FAST_MODE` limits, which is important for making a fair comparison. Settings with higher accuracy can make the quality of the solution even better, but they do so at a much higher cost in terms of computing power. This is something that will be worked on in the future.

6.5. Evaluation Metrics

Solver performance was assessed using a combination of optimization and application-level metrics:

- QUBO energy: the objective value returned by the Binary Quadratic Model, reported both with and without the constant offset;
- Runtime: wall-clock time to obtain the best solution;
- Time-to-solution trade-offs: relative speedups with respect to baseline SA;
- Decoded kinematic errors: camber and caster root mean square error (RMSE) with respect to the prescribed target curves.

These metrics allow us to disentangle purely algorithmic performance (objective value and runtime) from application-level behavior observed after decoding the binary solution back into continuous suspension geometry. The results of this evaluation are presented and discussed in Section 7.

7. Results

This section presents a comprehensive benchmark of the solver suite described in Section 5, applied to the QUBO instance constructed in Section 4. We first summarize global performance metrics across solvers, then analyze runtime–energy trade-offs, convergence behavior, and decoded kinematic outcomes. All results correspond to the experimental setup described in Section 6.

7.1. Global Solver Performance

Table 2 reports the best objective value, runtime, and decoded camber/caster errors for each solver. The QUBO energy is shown both as the raw solver output and with the constant offset added back for comparability.

Table 2. Solver performance summary on the suspension QUBO instance.Lower energy values indicate better objective quality. Camber and caster errors are reported as RMSE over the sampled roll range.

Solver	Energy (+offset)	Runtime (s)	Camber RMSE (deg)	Caster RMSE (deg)
SA	−837.03	12.92	15.25	6.91
Tabu	−763.63	52.48	18.39	6.91
Parallel-SA (4)	**−845.29**	26.15	15.25	6.91
Pop-SA (new C)	−826.05	6.19	15.25	6.91
BH-SA (new A)	−816.98	**0.30**	15.25	6.91
ADMM-QUBO	−768.38	**0.05**	15.54	**6.39**
ADMM-SA Hybrid (new B)	−796.53	0.05	18.39	6.91

a. Best energy across all solvers. *b*. Fastest runtime among SA-class and hybrid solvers, respectively. *c*. Best caster RMSE across all solvers. **Bold** entries denote the most competitive result for each respective solver.

Among all solvers, Parallel-SA achieves the lowest (best) objective value, improving upon baseline SA by approximately 8.3 energy units. However, this improvement comes at the cost of increased runtime due to multi-process overhead. By contrast, the hybrid and relaxation-based solvers achieve substantial reductions in wall-clock time, in some cases exceeding two orders of magnitude, while producing slightly higher objective values.

7.2. Runtime and Objective Trade-Offs

Figure 3 provides a visual comparison of solver runtime, objective value, and decoded camber/caster accuracy. Tabu Search is clearly dominated, exhibiting the longest runtime without commensurate gains in objective quality. Parallel-SA improves the objective relative to SA, while Pop-SA compromise between runtime and energy.

The Pareto-style plot in Figure 4 shows the trade-off between runtime and accuracy even more clearly. BH-SA and ADMM-QUBO are in the area with the lowest runtime, while Parallel-SA is in the area with the lowest energy but the highest runtime. This shows that no one solver is the best at everything.

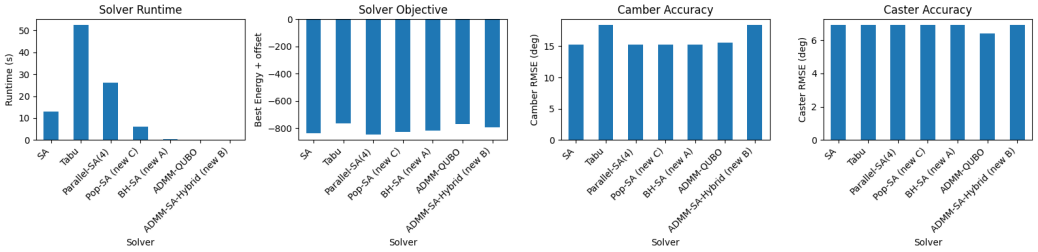

Figure 3. Bar plots comparing solver runtime, best objective value (energy + offset), camber RMSE, and caster RMSE. Parallel-SA achieves the lowest energy, while hybrid solvers provide substantial runtime reductions.

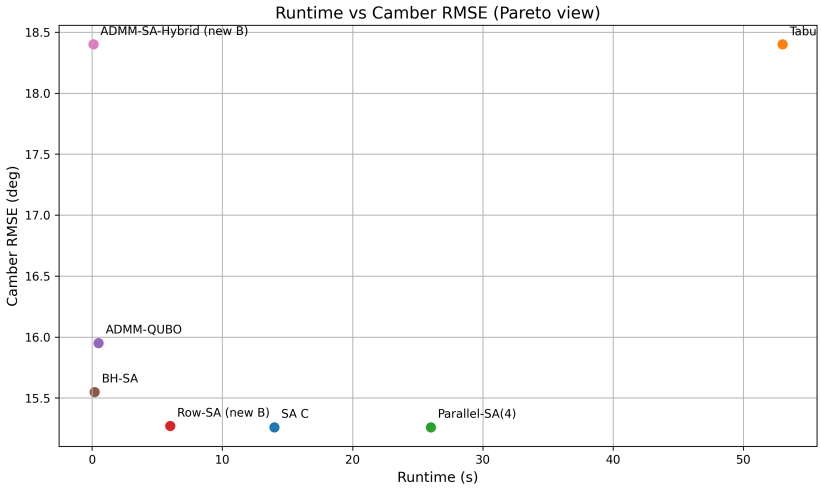

Figure 4. Runtime versus camber RMSE (Pareto view). Hybrid and relaxation-based solvers achieve orders-of-magnitude speedups, while Parallel-SA attains the best objective value at higher cost.

7.3. Relative Speedups and Baseline Comparison

We calculate solver speedups and energy differences to figure out how much better things are compared to baseline SA. Table 3 shows how well the solver performance compares to baseline SA.

Table 3. Solver performance relative to baseline SA. Positive ΔEnergy indicates worse objective value than SA.

Solver	Speedup vs. SA	ΔEnergy	ΔCamber RMSE
Parallel-SA (4)	0.49	−8.27	0.00
Pop-SA (new C)	2.09	+10.98	0.00
BH-SA (new A)	43.07	+20.05	0.00
ADMM-QUBO	271.06	+68.64	+0.29
ADMM-SA Hybrid	242.73	+40.49	+3.14

ADMM-based methods can speed things up a lot, but their objective values are always lower than SA's. BH-SA strikes a good balance by providing a speedup of more than 40× while keeping the same camber RMSE as SA.

7.4. Multi-Metric Dominance Analysis

To analyze solver performance across multiple metrics simultaneously, we normalize runtime, energy, camber RMSE, and caster RMSE to the range $[0, 1]$, with higher values indicating better performance. Figure 5 shows a dominance heatmap of the normalized scores.

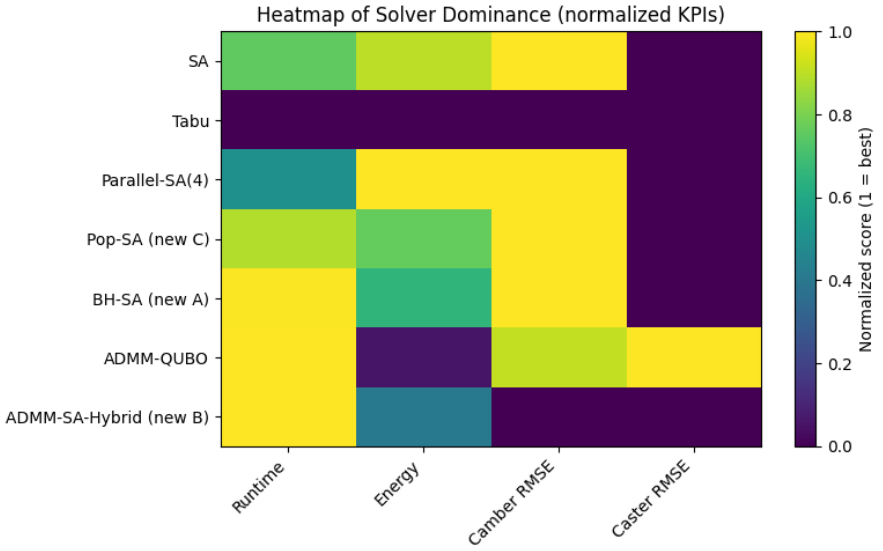

Figure 5. Heatmap of normalized solver dominance across runtime, energy, camber RMSE, and caster RMSE. No solver dominates all metrics simultaneously.

Figure 6 present the same data using parallel-coordinate and radar visualizations, respectively. These views emphasize the complementary strengths of different solver classes: energy-oriented (Parallel-SA), runtime-oriented (ADMM-QUBO), and balanced hybrids (BH-SA, Pop-SA).

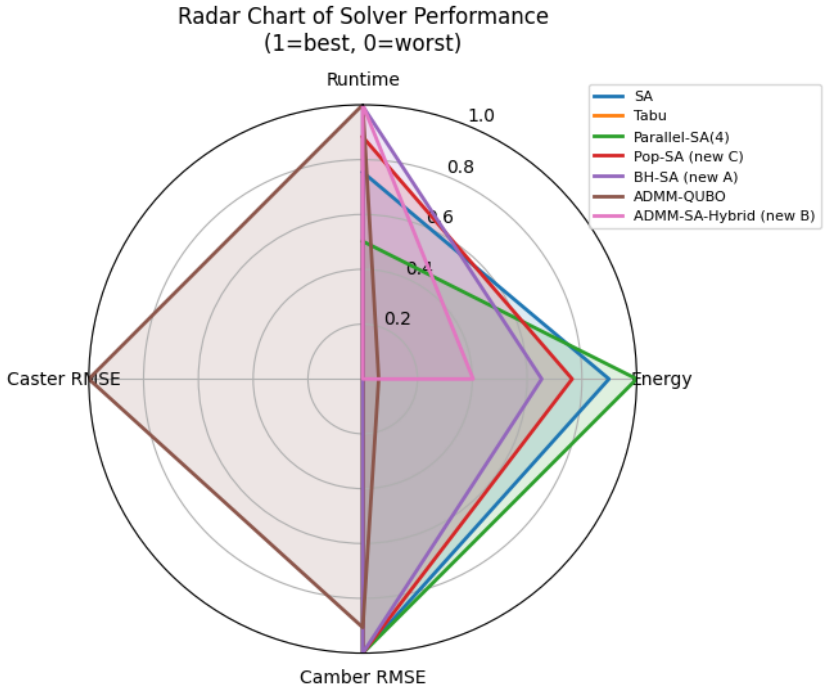

Figure 6. Radar chart of normalized solver performance. Larger filled areas indicate better overall trade-offs.

7.5. Convergence Behavior of ADMM-Based Methods

Figure 7 shows the energy evolution of the ADMM-QUBO solver and the ADMM phase of the ADMM–SA hybrid. Both exhibit rapid initial convergence followed by early stagnation, consistent with the role of ADMM as a fast relaxation rather than a global optimizer.

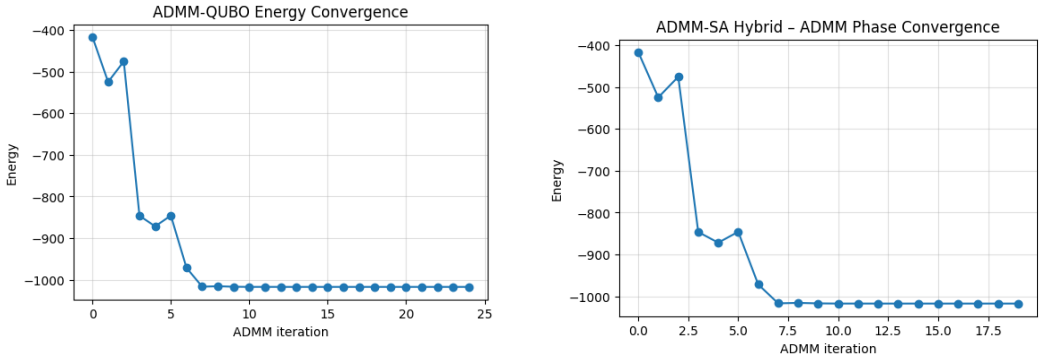

Figure 7. Convergence of ADMM-QUBO and ADMM–SA Hybrid. ADMM rapidly decreases the energy in the early iterations and then plateaus; the hybrid method applies SA refinement on top of the ADMM warm start.

7.6. Decoded Camber and Caster Curves

Although the primary focus of this work is solver benchmarking at the QUBO level, it is informative to examine the decoded camber and caster responses. Figure 8 compares target curves, baseline geometry, and selected solver outputs.

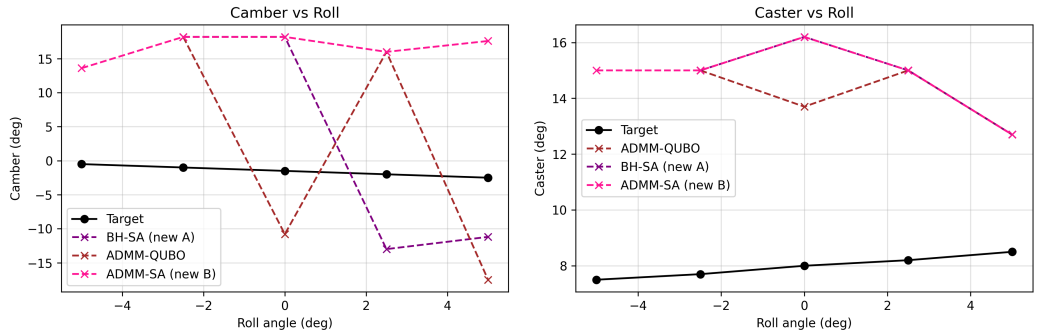

Figure 8. Camber and caster versus roll angle for selected solvers. All solvers improve caster RMSE relative to the baseline geometry, while camber tracking degrades due to the coarse binary discretization.

Across all solvers, camber RMSE increases substantially compared to the initial geometry, whereas caster RMSE improves modestly. This consistent pattern indicates that the low-resolution discretization ($B = 1$) and penalty structure dominate the decoded kinematic behavior, largely independent of the solver used.

7.7. Geometric Interpretation of Optimized Solutions

Finally, Figure 9 visualizes the baseline and optimized suspension geometries corresponding to the best-energy solution. The optimized configuration differs only subtly from the baseline, with millimeter-scale changes in hardpoint locations.

This observation reinforces the interpretation that the present study is best understood as a benchmark of solver behavior on a structured engineering-derived QUBO, rather than as a finalized suspension design exercise.

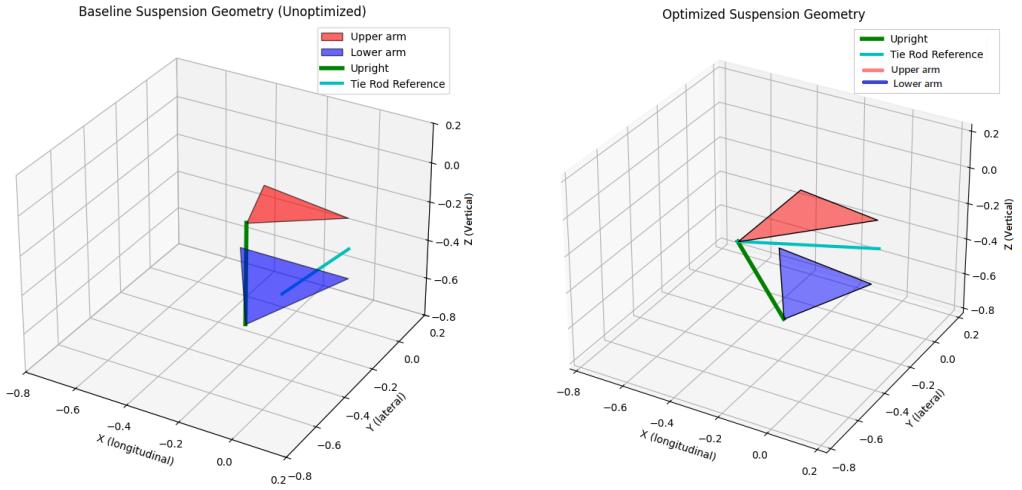

Figure 9. This shows the baseline suspension geometry on the left and the optimised suspension geometry on the right. Even though the QUBO objective has changed a lot, the decoded geometric differences are still small because the encoding is low-resolution.

8. Discussion

The results in Section 7 give us important information about how classical, parallel, and hybrid metaheuristic solvers work when they are used on a large QUBO instance that comes from a symbolic suspension kinematic model. Instead of focusing on the best possible design, this discussion looks at the trends that have been seen in terms of solver efficiency, objective landscape interaction, and modeling assumptions.

8.1. Objective Quality Versus Time-to-Solution

A key result of the benchmark is that it makes it clear that there is a trade-off between objective value and runtime. Parallel-SA consistently gets the lowest QUBO energy, which shows that running independent annealing increases the chances of finding low-energy basins. However, this improvement comes at a high cost in terms of computing power because it requires more annealing sweeps and more time to run in parallel.

In contrast, hybrid and relaxation-based solvers, especially BH-SA and ADMM-QUBO, cut wall-clock time by a huge amount, often by more than two orders of magnitude compared to baseline SA. These speedups come at the cost of higher objective values, which means that these methods tend to quickly find good but not globally competitive minima. From a practical point of view, this behaviour makes sense in situations where quick, rough solutions are better than a full search, like when you're exploring design ideas in the early stages or when you're working on interactive optimization.

8.2. Interpretation of Decoded Kinematic Performance

An important observation is that all solvers yield similar decoded camber and caster errors, despite noticeable differences in their QUBO objective values. In particular, the camber RMSE increases substantially relative to the baseline geometry, whereas the caster RMSE exhibits a modest improvement.

It is important to consider that this camber degradation interpreted as a modeling artifact rather than a limitation of the solvers themselves. The primary cause is the deliberately low-resolution binary encoding ($B = 1$), which restricts each continuous variable to only two admissible values. This coarse discretization severely limits the expressiveness of the design space and constrains the ability of any solver—regardless of quality—to finely adjust geometric parameters in order to match camber targets.

Additionally, the use of soft penalty terms and a coarse roll discretization leads to multiple binary configurations that satisfy constraint penalties similarly while producing comparable decoded kinematic responses. Under these modeling assumptions, improvements in the abstract QUBO objective do not necessarily translate into improvements in physical KPIs such as camber RMSE.

The consistent behavior across all solver classes reinforces the intended interpretation of this work: the benchmark primarily evaluates solver performance at the QUBO level under a fixed engineering-derived formulation. Decoded kinematic behavior is therefore included as a secondary diagnostic, not as an indicator of finalized suspension design quality.

8.3. Role of Hybrid and Adaptive Strategies

BH-SA stands out as a very good compromise among the hybrid approaches. It can quickly switch between SA and Tabu steps, which helps it make quick progress at the beginning of the search and avoid getting stuck for a long time. The performance that came out of it—over 40× faster than SA with no more camber RMSE degradation—shows how powerful simple learning-based solver orchestration can be, even when there are no formal convergence guarantees.

Similarly, ADMM-based methods demonstrate the value of continuous relaxations as fast approximators. Although ADMM-QUBO alone does not reach competitive energy levels, its rapid convergence makes it a useful component for warm-starting or screening large QUBO instances. The ADMM–SA hybrid partially recovers objective quality while retaining much of the speed advantage, illustrating how solver composition can be tailored to different time and accuracy budgets.

8.4. Implications for QUBO-Based Engineering Optimization

Taken together, the results suggest that solver choice should be guided by the intended role of the optimization. If the goal is to make the QUBO objective as small as possible with a fixed formulation, parallel annealing is still a good place to start. If, on the other hand, a quick turnaround is needed, hybrid and relaxation-based methods are good options.

More generally, the study shows that in engineering-derived QUBO problems, modeling choices like discretization resolution, penalty scaling, and surrogate accuracy often have a bigger effect on how the solver works later on. So, solver benchmarking should be seen in light of these modeling assumptions, not on its own.

All experiments were conducted using a low-resolution binary encoding ($B = 1$) to maintain a controlled and computationally tractable benchmark. Higher-resolution encodings ($B > 1$) are beyond the scope of the present work. However, increasing B would expand the binary dimensionality and typically increase interaction density and landscape ruggedness. In such settings, one would expect the qualitative solver trends observed here to persist and, in many cases, become more pronounced: parallel and hybrid methods would likely gain relative advantage due to increased search-space complexity, while relaxation-based methods might require more careful penalty calibration. Thus, higher-resolution encodings would likely amplify solver differentiation rather than invalidate the comparative conclusions drawn in this benchmark.

9. Conclusions

This paper presented a comprehensive computational benchmark of classical, parallel, and hybrid metaheuristic solvers applied to a QUBO-formulated suspension geometry optimization problem. Starting from a symbolic three-dimensional kinematic model of a double wishbone suspension, we constructed a large-scale QUBO instance using an explicit

low-resolution binary encoding and evaluated a diverse solver suite on a pre-exascale HPC platform.

The results demonstrate that no single solver dominates across all metrics. Parallel simulated annealing achieves the lowest objective values, while hybrid and relaxation-based methods deliver substantial reductions in time-to-solution. Adaptive and population-based strategies provide intermediate trade-offs that are attractive under constrained computational budgets. Across all solvers, decoded kinematic behavior remains largely governed by the discretization and penalty structure, underscoring the importance of modeling choices in QUBO-based engineering optimization. Rather than proposing a new suspension design methodology, this work contributes a reproducible benchmark and a detailed analysis of solver behavior on a structured, engineering-inspired QUBO. The findings offer practical guidance for selecting solvers based on runtime and objective trade-offs, and provide a reference point for future studies that aim to scale QUBO formulations, refine discretization schemes, or integrate higher-fidelity physical models.

Future work will focus on extending the framework to higher-resolution binary encodings, systematic sensitivity analysis of penalty weights, and multi-objective formulations that better balance competing kinematic targets. In addition, the same benchmarking methodology can be applied to other engineering design problems to further clarify the role of solver choice in large-scale QUBO optimization.

Author Contributions: Conceptualization, M.W.A. and S.L.; methodology, M.W.A. and S.L.; software, M.W.A.; formal analysis, M.W.A.; investigation, M.W.A.; data curation, M.W.A.; visualization, M.W.A.; writing—original draft preparation, M.W.A.; writing—review and editing, S.L., O.A., M.H.R. and S.R.H.; supervision, S.L.; project administration, S.L. and S.R.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Ministerial Decree No. 352 through [National Recovery and Resilience Plan (NRRP)] (funded by European Union's NextGenerationEU) under Mission 4 "Education and Research," Component 2 "From Research to Business," Investment 3.3, in collaboration with Ferrari S.p.A., under grant no. 2933.

Institutional Review Board Statement: Not Applicable.

Informed Consent Statement: Not Applicable.

Data Availability Statement: Data is contained within the article.

Acknowledgments: The authors would like to thank the Ferrari Technical Team—T. Davide, F. Alessandro, and F. Rocco—for all the help, technical advice, and interesting conversations they had with them while this research was being developed. Their knowledge was very important in deciding how to model things and proving that the proposed optimization framework was useful for engineering. The authors also thank CINECA in Italy for letting them use the HPC facility *Leonardo*, which had the computing power needed to do the large-scale optimization tests that are the subject of this study.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

QUBO	Quadratic Unconstrained Binary Optimization
HPC	High-Performance Computing
SA	Simulated Annealing
Parallel-SA	Parallel Simulated Annealing
Tabu	Tabu Search

BH-SA	Bandit-Hybrid Simulated Annealing–Tabu
Pop-SA	Population-Parallel Simulated Annealing
ADMM	Alternating Direction Method of Multipliers
ADMM-SA	ADMM–Simulated Annealing Hybrid
BQM	Binary Quadratic Model
UCA	Upper Control Arm
LCA	Lower Control Arm
RMSE	Root Mean Square Error
FAST_MODE	Reduced-runtime experimental configuration

Appendix A. Symbolic Kinematic Model Details

This appendix provides additional details on the symbolic suspension model used to construct the QUBO formulation described in Section 3. The purpose is to improve transparency while keeping the main manuscript focused on solver benchmarking.

Appendix A.1. Geometric Variables

The symbolic model includes the following geometric variables, each expressed in the vehicle-fixed coordinate frame $\{X, Y, Z\}$:

- Inboard joint locations: $\mathbf{o}_{\mathrm{UB}}$ (upper control arm), $\mathbf{o}_{\mathrm{LB}}$ (lower control arm), $\mathbf{o}_{\mathrm{SB}}$ (steering base),
- Relative link vectors: $\mathbf{p}_{1U}$ (UCA outboard direction), $\mathbf{o}_{\mathrm{HL}}$ (LCA outboard direction), $\mathbf{p}_{2S}$ (tie–rod direction),
- Auxiliary geometry vectors: $\mathbf{p}_{1H}$, $\mathbf{p}_{2H}$, $\mathbf{p}_{\mathrm{rear},U}$, and $\mathbf{p}_{\mathrm{rear},L}$.

Each vector contributes three scalar variables, resulting in a total of 30 continuous geometric degrees of freedom.

Appendix A.2. Polynomial Camber and Caster Surrogates

Camber and caster angles are obtained from the upright orientation and are originally defined through trigonometric relations. To enable QUBO construction, these expressions are expanded symbolically around the nominal geometry and small-angle operating conditions using low-order Taylor approximations.

The resulting surrogate models take the form

$$\hat{\gamma}^{(k)} = f_{\gamma}^{(k)}(\mathbf{x}, \boldsymbol{\Theta}), \qquad \hat{\kappa}^{(k)} = f_{\kappa}^{(k)}(\mathbf{x}, \boldsymbol{\Theta}), \tag{A1}$$

where $f_{\gamma}^{(k)}$ and $f_{\kappa}^{(k)}$ are multivariate polynomials evaluated at each roll sample k. These polynomial representations preserve smooth dependence on the design variables and remain compatible with binary encoding and quadratic reduction.

Appendix B. Solver Parameter Settings and FAST_MODE

This appendix summarizes the solver parameters used in the experiments reported in Section 7. All results correspond to the `FAST_MODE` configuration described in Section 6.4.

Appendix B.1. Baseline and Parallel Solvers

Table A1. Parameter settings for baseline and parallel solvers (`FAST_MODE`).

Solver	Key Parameters	Value
SA	Reads/sweeps	4000/800
Tabu	Reads	2000
Parallel-SA	Workers/total reads	4/8000

Appendix B.2. Hybrid and Population-Based Solvers

Table A2. Parameter settings for hybrid and population-based solvers (`FAST_MODE`).

Solver	Key Parameters	Value
BH-SA	Phases/SA sweeps	12/800
BH-SA	Tabu timeout/α	20 ms/0.3
Pop-SA	Population/generations	16/10
Pop-SA	SA sweeps per individual	400

Appendix B.3. ADMM-Based Solvers

Table A3. Parameter settings for ADMM-based solvers (`FAST_MODE`).

Solver	Key Parameters	Value
ADMM-QUBO	ADMM iterations/GD steps	25/4
ADMM-QUBO	Step size/ρ	0.05/1.0
ADMM–SA	ADMM iters/SA sweeps	20/800

All solvers use identical QUBO instances and fixed random seeds to ensure fair and reproducible comparisons.

Appendix C. HPC Configuration and Reproducibility Checklist

All experiments were performed on the CINECA Leonardo supercomputing system.

Appendix C.1. Hardware Environment

- Cluster type: EuroHPC pre-exascale system (Leonardo),
- CPU partition: 32-core AMD EPYC 7H12 nodes,
- Memory: 128 GB RAM per node,
- Operating system: Linux (CentOS),
- Scheduler: Slurm workload manager,
- Software environment: Python 3.11, Conda virtual environment.

Appendix C.2. Reproducibility Checklist

To facilitate reproducibility, the following practices were adopted:

- Fixed pseudo-random seeds for NumPy and all solvers,
- Identical QUBO instances used across all solver runs,
- Exclusive node allocation to avoid runtime interference,
- Explicit reporting of solver parameters and FAST_MODE settings,
- Consistent decoding and KPI evaluation pipeline.

All reported results can be reproduced by re-running the experiments under the same configuration.

References

1. Gillespie, T. *Fundamentals of Vehicle Dynamics*; SAE International: Warrendale, PA, USA, 2021. Available online: https://www.sae.org/books/fundamentals-vehicle-dynamics-revised-edition-r-506 (accessed on 21 December 2025).
2. Hu, S.; Rui, X.; Gu, J.; Zhang, X. Dynamics Modeling and Suspension Parameters Optimization of Vehicle System Based on Reduced Multibody System Transfer Matrix Method. *Machines* **2025**, *13*, 116. [CrossRef]
3. Lucas, A. Ising formulations of many NP problems. *Front. Phys.* **2014**, *2*, 5 . [CrossRef]
4. Campos, R. Hybrid Quantum-Classical Algorithms. *arXiv* **2024**, arXiv:2406.12371. [PubMed]
5. Vandelli, M.; Ferrari, F.; Dragoni, D. Parallel splitting method for large-scale quadratic programs. *arXiv* **2025**, arXiv:2503.16977. [CrossRef]

6. Kong, S.; Ye, Y.; Wu, Y.; Wang, S.; Hou, J.; Ni, M. Solving Combinatorial Optimization Problems Based on QUBO Models. In *Proceedings of the 2024 4th International Symposium on Computer Technology and Information Science (ISCTIS)*; IEEE: Piscataway, NJ, USA, 2024. [CrossRef]
7. Nasr, M.; Farouk, O.; Mohamedeen, A.; Elrafie, A.; Bedeir, M.; Khaled, A. Benchmarking Meta-heuristic Optimization. *Int. J. Adv. Netw. Appl.* **2020**, *11*, 4451–4457. [CrossRef]
8. Said, G.A.E.-N.A.; Mahmoud, A.M.; El-Horbaty, E.-S.M. A Comparative Study of Meta-heuristic Algorithms for Solving Quadratic Assignment Problem. *Int. J. Adv. Comput. Sci. Appl.* **2014**, *5*, 1–5. [CrossRef]
9. Goto, H.; Endo, K.; Suzuki, M.; Sakai, Y.; Kanao, T.; Hamakawa, Y.; Hidaka, R.; Yamasaki, M.; Tatsumura, K. High-performance combinatorial optimization based on classical mechanics. *Sci. Adv.* **2021**, *7*, eabe7953. [CrossRef] [PubMed]
10. Pavlukhin, P.; Menshov, I. On implementation high-scalable CFD solvers for hybrid clusters with massively-parallel architectures. In *Proceedings of the Parallel Computing Technologies (PaCT 2015)*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2015; Volume 9251, pp. 480–491. [CrossRef]
11. Huang, M.W.; Arora, J. Engineering optimization with discrete variables. In *Proceedings of the 36th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*; AIAA: Reston, VA, USA, 1995; Volume 3, pp. 1475–1485. [CrossRef]
12. Matsumori, T.; Taki, M.; Kadowaki, T. Application of QUBO solver using black-box optimization to structural design for resonance avoidance. *Sci. Rep.* **2022**, *12*, 12143. [CrossRef] [PubMed]
13. Arshad, M.W.; Lodi, S.; Liu, D.Q. An Advanced Hybrid Optimization Algorithm for Vehicle Suspension Design Using a QUBO-SQP Framework. *Mathematics* **2025**, *13*, 3843. [CrossRef]
14. Maciejewski, F.B.; Bach, B.G.; Dupont, M.; Lott, P.A.; Sundar, B.; Neira, D.E.B.; Safro, I.; Venturelli, D. A Multilevel Approach For Solving Large-Scale QUBO Problems With Noisy Hybrid Quantum Approximate Optimization. *arXiv* **2024**, arXiv:2408.07793. [CrossRef]
15. Llopis-Albert, C.; Rubio, F.; Zeng, S. Multiobjective optimization framework for designing a vehicle suspension system. A comparison of optimization algorithms. *Adv. Eng. Softw.* **2023**, *176*, 103375. [CrossRef]
16. Kodati, M.; Bandyopadhyay, S. Kinematic analysis of the double wishbone suspension system. In *Proceedings of the 1st International and 16th National Conference on Machines and Mechanisms (iNaCoMM 2013)*; IIT Roorkee: Roorkee, India, 2013; pp. 562–567. Available online: https://api.semanticscholar.org/CorpusID:53489911 (accessed on 25 December 2025).
17. Arshad, M.W.; Lodi, S.; Liu, D.Q. Double Wishbone Suspension: A Computational Framework for Parametric 3D Kinematic Modeling and Simulation Using Mathematica. *Technologies* **2025**, *13*, 332. [CrossRef]
18. Liu, J. Optimal design and analysis of intelligent vehicle suspension system based on ADAMS and artificial intelligence algorithms. *J. Phys. Conf. Ser.* **2021**, *2074*, 012023. [CrossRef]
19. Ajay Ganesh Ram, V.; Easwar, T.; Vishal, A.; Andrews, A.; Michael Thomas Rex, F. Design, optimization and analysis of baja suspension system using full factorial approach. In *Advances in Materials and Manufacturing Engineering: Select Proceedings of ICMME 2019*; Rajmohan, T., Palanikumar, K., Davim, J.P., Eds.; Springer Proceedings in Materials; Springer: Singapore, 2021; Volume 7, pp. 23–29. [CrossRef]
20. Belluomo, C.; Lenzo, B.; Bucchi, F.; Velardocchia, M. Design, analysis and investigation of an independent suspension for passenger cars. In *Advances in Italian Mechanism Science. IFToMM ITALY 2018*; Rossi, C., Mejia, K., Eds.; Mechanisms and Machine Science; Springer: Cham, Switzerland, 2019; Volume 68, pp. 165–173. [CrossRef]
21. Arshad, M.W.; Lodi, S. Quantum computing in the automotive industry: Survey, challenges, and perspectives. *J. Supercomput.* **2025**, *81*, 1093. [CrossRef]
22. Yarkoni, S.; Raponi, E.; Bäck, T.; Schmitt, S. Quantum annealing for industry applications: Introduction and review. *Rep. Prog. Phys.* **2022**, *85*, 104001. [CrossRef] [PubMed]
23. D-Wave Systems Inc. *D-Wave Ocean Dwave-Samplers Documentation: SimulatedAnnealingSampler*; Documentation; D-Wave Systems Inc.: Burnaby, BC, Canada, 2025. Available online: https://docs.dwavequantum.com/en/latest/ocean/api_ref_samplers/generated/dwave.samplers.SimulatedAnnealingSampler.sample.html (accessed on 10 January 2025).
24. D-Wave Systems Inc. *D-Wave Ocean Dwave-Tabu Documentation: TabuSampler*; Documentation; D-Wave Systems Inc.: Burnaby, BC, Canada, 2025. Available online: https://docs.dwavequantum.com/en/latest/ocean/api_ref_samplers/generated/dwave.samplers.TabuSampler.sample.html (accessed on 10 January 2025).
25. D-Wave Systems Inc. *D-Wave Ocean Hybrid Documentation: Kerberos (KerberosSampler) Reference Workflow*; Documentation; D-Wave Systems Inc.: Burnaby, BC, Canada, 2025. Available online: https://docs.dwavequantum.com/en/latest/ocean/api_ref_hybrid/reference.html (accessed on 10 January 2025).

26. Bowles, J.; Dauphin, A.; Huembeli, P.; Martinez, J.; Acín, A. Quadratic Unconstrained Binary Optimization via Quantum-Inspired Annealing. *Phys. Rev. Appl.* **2022**, *18*, 034016. [CrossRef]
27. D-Wave Systems Inc. *D-Wave Ocean Dimod Documentation: Binary Quadratic Models and BinaryQuadraticModel*; Documentation; D-Wave Systems Inc.: Burnaby, BC, Canada, 2025. Available online: https://docs.dwavequantum.com/en/latest/ocean/api_ref_dimod/ (accessed on 12 January 2025).