Fast and Robust Constrained Optimization via Evolutionary and Quadratic Programming

Konstantinos I. Chatzilygeroudis(✉) and Michael N. Vrahatis

Computational Intelligence Laboratory (CILab), Department of Mathematics, University of Patras, 26110 Patras, Greece
costashatz@upatras.gr, vrahatis@math.upatras.gr

Abstract. Many efficient and effective approaches have been proposed in the evolutionary computation literature for solving constrained optimization problems. Most of the approaches assume that both the objective function and the constraints are black-box functions, while a few of them can take advantage of the gradient information. On the other hand, when the gradient information is available, the most versatile approaches are arguably the ones coming from the numerical optimization literature. Perhaps the most popular methods in this field are sequential quadratic programming and interior point. Despite their success, those methods require accurate gradients and usually require a well-shaped initialization to work as expected. In the paper at hand, a novel hybrid method, named UPSO-QP, is presented that is based on particle swarm optimization and borrows ideas from the numerical optimization literature and sequential quadratic programming approaches. The proposed method is evaluated on numerous constrained optimization tasks from simple low dimensional problems to high dimensional realistic trajectory optimization scenarios, and showcase that is able to outperform other evolutionary algorithms both in terms of convergence speed as well as performance, while also being robust to noisy gradients and bad initialization.

Keywords: Constrained Optimization · Particle Swarm Optimization · Quadratic Programming

1 Introduction and Related Work

Constraint Optimization Problems (COP) appear in many diverse research fields and applications, including, among others, structural optimization, engineering design, VLSI design, economics, allocation and location problems, robotics and optimal control problems [9,11,17,28]. All these real-world problems are typically represented by a mathematical model that can contain both binary and

This work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "3rd Call for H.F.R.I. Research Projects to support Post-Doctoral Researchers" (Project Acronym: NOSALRO, Project Number: 7541).

M. Sellmann and K. Tierney (Eds.): LION 2023, LNCS 14286, pp. 46–61, 2023.
https://doi.org/10.1007/978-3-031-44505-7_4

continuous variables, and a set of linear and non-linear constraints, ranging from simple, low dimensional numerical functions to high-dimensional noisy estimates.

The *Numerical Optimization* (NumOpt) literature [11,17] has given us a wide range of powerful tools to tackle those problems. Methods such as *Feasible Direction* (FD) [2], *Generalized Gradient Descent* (GGD) [18], *Interior Point* (IP) [27] and *Sequential Quadratic Programming* (SQP) [17] are able to solve very effectively COP problems even in high dimensions as well as to tackle problems for cases where the objective function or any of the constraints are non-convex. Despite their success, those methods require the availability of the exact gradient values of both the objective and constraints functions, which can be difficult to have in several real-world situations where only approximations are available. Moreover, it is well known that in practice those methods require good initialization to achieve reasonable convergence rates. On the other hand, many *Evolutionary Algorithms* (EAs) have been proposed for solving COP problems [4–7,12,21,24] and have been applied with success to many COP problems, even in cases where the corresponding function values are corrupted by noise. Most of the methods do not require the knowledge of the gradients [12,21], while some of them attempt to improve convergence or performance by using the gradient information [5]. Nevertheless, EAs are known to need many functions evaluations to be able to find high performing solutions, and can often fail to find the global optimum.

In the paper at hand, we take inspiration both from the numerical optimization and the evolutionary computation literature and propose UPSO-QP, a novel approach for solving COPs that attempts to merge the two fields. In particular, UPSO-QP is based on *Particle Swarm Optimization* (PSO) [20,22] and borrows some ideas from SQP. We extensively evaluate UPSO-QP in many scenarios ranging from low-dimensional noiseless settings to high-dimensional non-convex problems, and showcase that UPSO-QP is able to outperform other evolutionary algorithms both in terms of convergence speed as well as performance, while also being robust to noisy gradients and bad initialization.

The rest of the paper is organized as follows. In Sect. 2 the problem formulation and a brief presentation of the required background material are presented. In Sect. 3 a detailed description of the proposed method is provided, while in Sect. 4 experimental results are presented. The paper ends in Sect. 5 with some concluding remarks.

2 Problem Formulation and Background Material

We aim at solving the following problem:

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^N}{\operatorname{argmin}} f(\boldsymbol{x}), \\ &\text{s.t.} \quad h_i(\boldsymbol{x}) = 0, \\ &\qquad\quad g_j(\boldsymbol{x}) \geqslant 0, \end{aligned} \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^N$ is the optimization variable, $f : \mathbb{R}^N \to \mathbb{R}$ is the objective function, $h_i : \mathbb{R}^N \to \mathbb{R}$ with $i = 1, 2, \ldots, N_{\text{eq}}$ are the equality constraints, and $g_j : \mathbb{R}^N \to \mathbb{R}$ with $j = 1, 2, \ldots, N_{\text{ineq}}$ are the inequality constraints.

2.1 Particle Swarm Optimization

Particle Swarm Optimizers (PSO) [20,22] are evolutionary algorithms that are inspired by the aggregating behaviors of populations. PSO algorithms consist of a swarm of particles that are randomly positioned in the search space and communicate with their neighbors. Each particle performs objective function evaluations and updates its position as a function of previous evaluations of its neighborhood and the whole swarm. The two main strategies are *local PSO* (PSO-LS) and *global PSO* (PSO-GS). In the local strategy, each particle has memory of the its best position ever visited, and the single best position ever visited by its neighbors. In the global strategy, particles maintain memory of their personal best position and the best position ever found by the whole swarm.

More formally, a predetermined set of M particles are initialized at random positions in the search space. The position of particle q at initialization (first iteration, $k = 0$) is denoted by $\boldsymbol{x}_q(0) \in \mathbb{R}^N$. Additionally, each particle is provided with a random velocity vector denoted $\boldsymbol{v}_q(0) \in \mathbb{R}^N$. At each iteration $k+1$, particles update their positions with the following equations:

$$\boldsymbol{x}_q(k+1) = \boldsymbol{x}_q(k) + \boldsymbol{v}_q(k+1). \tag{2}$$

In the *local PSO strategy*, the velocity is given by $\boldsymbol{l}_q(k+1) = \chi\left[\boldsymbol{v}_q(k) + c_1 r_1\left(\boldsymbol{x}_q^b(k) - \boldsymbol{x}_q(k)\right) + c_2 r_2\left(\boldsymbol{x}_q^{lb}(k) - \boldsymbol{x}_q(k)\right)\right]$, while in the *global PSO strategy* $\boldsymbol{g}_q(k+1) = \chi\left[\boldsymbol{v}_q(k) + c_1 r_1\left(\boldsymbol{x}_q^b(k) - \boldsymbol{x}_q(k)\right) + c_2 r_2\left(\boldsymbol{x}^b(k) - \boldsymbol{x}_q(k)\right)\right]$. $\boldsymbol{x}_q^b(k)$ denotes the best position visited by particle i from initialization and up to time k, and similarly, $\boldsymbol{x}_q^{lb}(k)$ and $\boldsymbol{x}^b(k)$ denote the best position ever found by the neighbors of i and the whole swarm respectively. Scalars c_1, c_2 are user defined parameters and $r_1, r_2 \in [0, 1]$ are randomly generated numbers, while χ is a user defined parameter similar to what learning rate is in gradient descent. The *Unified Particle Swarm Optimizer* (UPSO) [21] is an algorithm that combines the behaviors of the local and global PSO strategies:

$$\boldsymbol{v}_q(k+1) = u\,\boldsymbol{g}_q(k+1) + (1-u)\,\boldsymbol{l}_q(k+1), \tag{3}$$

where $u \in [0, 1]$ is a user defined parameter. Notice that if $u = 0$, UPSO coincides with the local strategy, PSO-LS, whereas, if $u = 1$, UPSO coincides with PSO-GS. In that sense, UPSO gives the "best of both worlds" by allowing the user to achieve superior performance with the fine-tuning of a single parameter.

2.2 Sequential Linear Quadratic Programming

The main intuition of *Sequential Linear Quadratic Programming* (SLQP) is to tackle the problem of Eq. (1) by splitting it into easier subproblems that are iteratively solved. In particular, the problem at each iteration is split into to two phases: a) the *Linear Programming* (LP) phase, and b) the *Equality Quadratic Programming* (EQP) phase. Before delving more into the details of SLQP, we can first see that the Lagrangian of Eq. (1) is as follows:

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= f(\boldsymbol{x}) - \sum_i \lambda_i h_i(\boldsymbol{x}) - \sum_j \mu_j g_j(\boldsymbol{x}) \\ &= f(\boldsymbol{x}) - \boldsymbol{\lambda}^\top \boldsymbol{h}(\boldsymbol{x}) - \boldsymbol{\mu}^\top \boldsymbol{g}(\boldsymbol{x}),\end{aligned} \tag{4}$$

where $\boldsymbol{h}(\cdot)$ and $\boldsymbol{g}(\cdot)$ are the stacked versions of the constraints.

In the first phase of SLQP, the problem is linearized around the current estimate $\boldsymbol{x}_k$ and an LP is formulated as follows:

$$\begin{aligned}&\underset{\boldsymbol{p}\in\mathbb{R}^N}{\operatorname{argmin}} f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^\top \boldsymbol{p}, \\ &\text{s.t.} \quad h_i(\boldsymbol{x}_k) + \nabla h_i(\boldsymbol{x}_k)^\top \boldsymbol{p} = 0, \\ &\qquad g_j(\boldsymbol{x}_k) + \nabla g_j(\boldsymbol{x}_k)^\top \boldsymbol{p} \geqslant 0, \\ &\qquad \|\boldsymbol{p}\|_\infty \leqslant \Delta_k^{\mathrm{LP}},\end{aligned} \tag{5}$$

where $f(\boldsymbol{x}_k)$ can be omitted from the optimization since it is constant, the solution of the problem is defined as $\boldsymbol{x}_k^{\mathrm{LP}} = \boldsymbol{x}_k + \boldsymbol{p}^{\mathrm{LP}}$, and Δ_k^{LP} is a trust-region radius in order to make the problem bounded.

Once the above problem is solved, we define the *Active Sets*, $\mathcal{A}_k^{\mathrm{eq}}$ and $\mathcal{A}_k^{\mathrm{ineq}}$, and the *Violating Sets*, $\mathcal{V}_k^{\mathrm{eq}}$ and $\mathcal{V}_k^{\mathrm{ineq}}$ to be the sets where the constraints are equal to zero and where the constraints are violated respectively.

In the second phase of SLQP, we define the following EQP problem:

$$\begin{aligned}\underset{\boldsymbol{p}\in\mathbb{R}^N}{\operatorname{argmin}}\, f(\boldsymbol{x}_k) + \frac{1}{2}\boldsymbol{p}^\top \nabla^2_{\boldsymbol{xx}} \mathcal{L}_k\, \boldsymbol{p} + \Big(\nabla f(\boldsymbol{x}_k) + \alpha_k \sum_{i\in\mathcal{V}_k^{\mathrm{eq}}} \gamma_i \nabla h_i(\boldsymbol{x}_k) \\ + \alpha_k \sum_{j\in\mathcal{V}_k^{\mathrm{ineq}}} \gamma_j \nabla g_j(\boldsymbol{x}_k)\Big)^\top \boldsymbol{p},\end{aligned}$$

$$\begin{aligned}&\text{s.t.} \quad h_i(\boldsymbol{x}_k) + \nabla h_i(\boldsymbol{x}_k)^\top \boldsymbol{p} = 0, i \in \mathcal{A}_k^{\mathrm{eq}}, \\ &\qquad g_j(\boldsymbol{x}_k) + \nabla g_j(\boldsymbol{x}_k)^\top \boldsymbol{p} = 0, j \in \mathcal{A}_k^{\mathrm{ineq}}, \\ &\qquad \|\boldsymbol{p}\|_2 \leqslant \Delta_k^{\mathrm{EQP}},\end{aligned} \tag{6}$$

where $\nabla^2_{\boldsymbol{xx}}\mathcal{L}_k$ is the Hessian of the Lagrangian over the optimization variables $\boldsymbol{x}$ evaluated at the current estimate $(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k)$, γ_i, γ_j are the algebraic signs of the i-th or j-th violated constraint, α_k is a penalty factor, and Δ_k^{EQP} is a trust-region radius in order to make the problem bounded. Practical implementations include line search, techniques for updating the penalty factors, estimating the Hessian instead of computing it, and trust-region radii as well as introducing slack variables to make the sub-problems always feasible (linearization can yield infeasible problems). For more details, we refer the interested reader to [17] and the references therein.

3 The Proposed UPSO-QP Approach

In the paper at hand, we combine the *Unified PSO* (UPSO) with *Sequential Linear Quadratic Programming* (SLQP). The intuition lies in the fact that SLQP is among the "strongest" nonlinear optimizers in the literature and practical applications, while UPSO is effective in black-box settings including constrained optimization [19,21]. The goal of our approach is to "fuse" the robustness and ease of usage of PSO methods with the convergence properties of SLQP methods. To this end, we propose a new hybrid algorithm, called UPSO-QP, that is based on UPSO, but also borrows ideas from SLQP. UPSO-QP follows the general UPSO framework, but we make some alternations to greatly improve its convergence when gradient (possibly imprecise or noisy) information is available.

3.1 Local QP Problems

First, we add a procedure to take advantage of gradient information of the objective and constraint functions. In particular, each particle q with probability $r_{\text{qp}} \in [0, 1]$ will solve the following QP problem:

$$\begin{aligned} \underset{\boldsymbol{p} \in \mathbb{R}^N}{\operatorname{argmin}} \; & \frac{1}{2} \boldsymbol{p}^\top \boldsymbol{p} + \nabla f\big(\boldsymbol{x}_q(k)\big)^\top \boldsymbol{p}, \\ \text{s.t.} \quad & h_i\big(\boldsymbol{x}_q(k)\big) + \nabla h_i\big(\boldsymbol{x}_q(k)\big)^\top \boldsymbol{p} = 0, \\ & g_j\big(\boldsymbol{x}_q(k)\big) + \nabla g_j\big(\boldsymbol{x}_q(k)\big)^\top \boldsymbol{p} \geqslant 0, \\ & \|\boldsymbol{p}\|_\infty \leqslant v_{\max}, \end{aligned} \tag{7}$$

where $v_{\max}$ is the maximum allowed velocity for each particle. This problem is inspired by the LP phase of SLQP (and in general by the SQP literature) with the added quadratic cost. This problem, similar to Eq. (5), can be infeasible because of the linearization. Instead of adding slack variables to ensure the feasibility of the problem (or other similar "tricks" from the numerical optimization literature), we take a practical approach, give the QP solver a fixed iteration budget and take the solution it has achieved so far even if infeasible. If the problem is infeasible, most QP solvers will converge to the least squares solution of the problem. So we expect to get a least squares approximation if the linearization yields infeasibility. In any case, we assume the solution returned by the QP problem to be $\boldsymbol{v}_q^{\text{qp}}(k+1)$, while we denote the update from UPSO as $\boldsymbol{v}_q^{\text{pso}}(k+1)$. The final velocity for each particle q is computed as:

$$\boldsymbol{v}_q(k+1) = \alpha_{\text{qp}}\, \boldsymbol{v}_q^{\text{qp}}(k+1) + \big(1 - \alpha_{\text{qp}}\big)\, \boldsymbol{v}_q^{\text{pso}}(k+1), \tag{8}$$

for a user defined parameter $\alpha_{\text{qp}} \in [0, 1]$. In this paper, we use the ProxQP solver [1] to solve the QP problems.

3.2 UPSO for Constrained Optimization

Apart from moving into the "right" direction, we also need a method for comparing particles. This is important since the "best" particle (either in the neighborhood or globally) is crucial for UPSO's performance. We follow [19] and we augment the objective function with a penalty function:

$$\tilde{f}(\boldsymbol{x}) = f(\boldsymbol{x}) + H(\boldsymbol{x}), \tag{9}$$

where

$$\begin{aligned} H(\boldsymbol{x}) &= h(k)\, P(\boldsymbol{x}), \\ P(\boldsymbol{x}) &= \sum_i \theta\big(\mathrm{cv}_i(\boldsymbol{x})\big)\, \mathrm{cv}_i(\boldsymbol{x})^{\gamma(\mathrm{cv}_i(\boldsymbol{x}))} + \sum_j \theta\big(\mathrm{cv}_j(\boldsymbol{x})\big)\, \mathrm{cv}_j(\boldsymbol{x})^{\gamma(\mathrm{cv}_j(\boldsymbol{x}))}, \\ \mathrm{cv}_i(\boldsymbol{x}) &= \big|h_i(\boldsymbol{x})\big|, \\ \mathrm{cv}_j(\boldsymbol{x}) &= \big|\min\big\{0,\, g_j(\boldsymbol{x})\big\}\big|, \\ h(k) &= k\sqrt{k}. \end{aligned} \tag{10}$$

We use the same functions $\theta(\cdot)$ and $\gamma(\cdot)$ as in [19].

3.3 Considerations

The main idea is that solving the problem in Eq. (5) will push each particle to follow the local linearized approximation of the original problem. This approximation is very effective close to the actual solution, while it can be bad in far away regions. The main intuition behind this merging of UPSO with SLQP is that this local approximation will generally move the particles closer to the solution, while UPSO can compensate for inaccuracies of those approximations. Moreover, this problem is solved individually by each particle and thus solved in many different locations of the search space simultaneously. In this way, we increase the probability that one of the initial conditions will be in a good region to enable convergence to the global solution. Moreoever, we get an *implicit averaging* [13,25] effect that helps UPSO "see through" the noise and inaccuracies and converge to a better optimum. Additionally, the constraint $\|\boldsymbol{p}\|_\infty \leqslant v_{\max}$ in Eq. (5) ensures that we stay in regions where the local linearization is expected to be true, and thus produce well behaved search velocities.

In smooth and well-behaved objective and constraint functions, the solution of Eq. (5) will provide strong directions towards the optimal solution, and thus accelerating the convergence of UPSO. In noisy, discontinuous and/or non-convex problems, the solution of Eq. (5) will at least provide an approximate direction towards minimizing the constraint violation (least squares solution) that can help UPSO converge faster.

The parameter r_{qp} is used to handle the trade-off between effectiveness and wall time performance. The bigger the value more particles will solve the QP

and thus we get better approximations of the search landscape. At the same time, this means that we solve more QP problems that can increase the wall time significantly. On the other hand, the parameter α_{qp} is used to specify how much we want to trust the solution of the QP problem. In smooth and noiseless functions, we should set $\alpha_{\text{qp}} \approx 1$ since the solution of the QP will most likely provide a good search direction. On the contrary, in noisy or non-convex problems we should decrease this value, as the QP estimate can be less accurate and even misleading. Overall, one can change the behavior of the solver by setting the appropriate values to these parameters.

4 Experiments

We extensively evaluate the effectiveness of UPSO-QP with multiple experiments and comparing to strong baselines. We aim at answering the following questions:

a) How does UPSO-QP perform in well-defined numerical constrained optimization problems? How does it compare to other evolutionary algorithms? How does it compare to state of the art SQP and IPM methods? We will answer those questions in Sect. 4.1.
b) How does UPSO-QP handle problems with noisy values and gradients? How does it compare in this domain to state of the art SQP and IPM methods? We will answer those questions in Sect. 4.2.
c) How does UPSO-QP operate on realistic high-dimensional constrained optimization problems? How sensitive it is in well-shaped initialization? How does it compare to state of the art SQP and IPM methods? We will answer those questions in Sect. 4.3.

In the subsequent sections, we compare the following algorithms:

1) UPSO-QP — custom implementation in C++ of our approach[1].
2) UPSO augmented with a penalty function for constrained optimization as in [19] (UPSO-Pen)—we use our own custom C++ implementation.
3) UPSO augmented with a penalty function and gradient-based repair technique as in [5] (UPSO-Grad) — we use our own custom C++ implementation.
4) Sequential Least Squares Programming (SLSQP) — this is an SQP approach as implemented in [14] and it is closely related to SLQP as described above[2].
5) A Primal-Dual Interior Point Algorithm as described in [27] (Ipopt) — this is a state-of-the-art IPM method widely used in practice[3].

We have carefully chosen the algorithms to compare UPSO-QP to in order to be able to highlight the main properties of our proposed method. In particular, UPSO-Pen does not have access to any gradient information and the penalty function technique is one of the most widely used in the evolutionary computation community. UPSO-Grad is an evolutionary method that takes advantage

[1] The code is available at https://github.com/NOSALRO/algevo.
[2] We use the implementation provided by scipy.
[3] We use the C++ implementation provided by the Ipopt library.

of the gradient information of the constraints in order to improve performance and has been shown to have superior performance over other evolutionary techniques [3]. Lastly, SLSQP and Ipopt are two of the most versatile and widely used numerical optimization algorithms.

4.1 Numerical Constrained Optimization Problems

In the first set of experiments, we select two low-dimensional numerical constrained optimization problems with known optimal solutions. This will give us the ability to extensively test and compare UPSO-QP with other methods, both from the evolutionary computation literature as well as the numerical optimization one. For each problem/algorithm pair we run 20 replicates with different initial conditions. For all the evolutionary algorithms, we used $M = 40$ particles, with 10 neighborhoods, $\chi = 0.729, c_1 = c_2 = 2.05$ and $u = 0.5$. For UPSO-QP, we also set $r_{\text{qp}} = 0.5$ and $\alpha_{\text{qp}} = 1$.

Problem 1. For $f : \mathbb{R}^2 \to \mathbb{R}$ [11],

$$\begin{aligned} f(\boldsymbol{x}) &= (x_1 - 2)^2 + (x_2 - 1)^2, \\ h_1(\boldsymbol{x}) &= x_1 - 2\,x_2 + 1 = 0, \\ g_1(\boldsymbol{x}) &= -0.25\,x_1^2 - x_2^2 + 1 \geqslant 0. \end{aligned}$$

The best known optimal feasible solution is $f(\boldsymbol{x}^*) = 1.3934651$.

Problem 2. For $f : \mathbb{R}^6 \to \mathbb{R}$ [15],

$$f(\boldsymbol{x}) = -10.5\,x_1 - 7.5\,x_2 - 3.5\,x_3 - 2.5\,x_4 - 1.5\,x_5 - 10\,x_6 - 0.5 \sum_{i=1}^{5} x_i^2,$$

$$\begin{aligned} g_i(\boldsymbol{x}) &= x_i \geqslant 0, \;\; i = 1, 2, \ldots, 5, \\ g_{i+5}(\boldsymbol{x}) &= 1 - x_i \geqslant 0, \;\; i = 1, 2, \ldots, 5, \\ g_{11}(\boldsymbol{x}) &= x_6 \geqslant 0, \\ g_{12}(\boldsymbol{x}) &= 6.5 - 6\,x_1 - 3\,x_2 - 3\,x_3 - 2\,x_4 - x_5 \geqslant 0, \\ g_{13}(\boldsymbol{x}) &= 20 - 10\,x_1 - 10\,x_3 - x_6 \geqslant 0. \end{aligned}$$

The best known optimal feasible solution is $f(\boldsymbol{x}^*) = -213$.

Results. The results showcase that UPSO-QP outperforms all other evolutionary baselines and always converges to the optimal solution faster (*cf.* Fig. 1, 2). Moreover compared to SLSQP and IPopt, UPSO-QP is able to achieve the same level of accuracy while also having comparable total wall time measurements.

4.2 Constrained Optimization with Noisy Functions Values

In this section, we will solve the same problems as above but we will add noise in different ways to showcase the robustness of UPSO-QP.

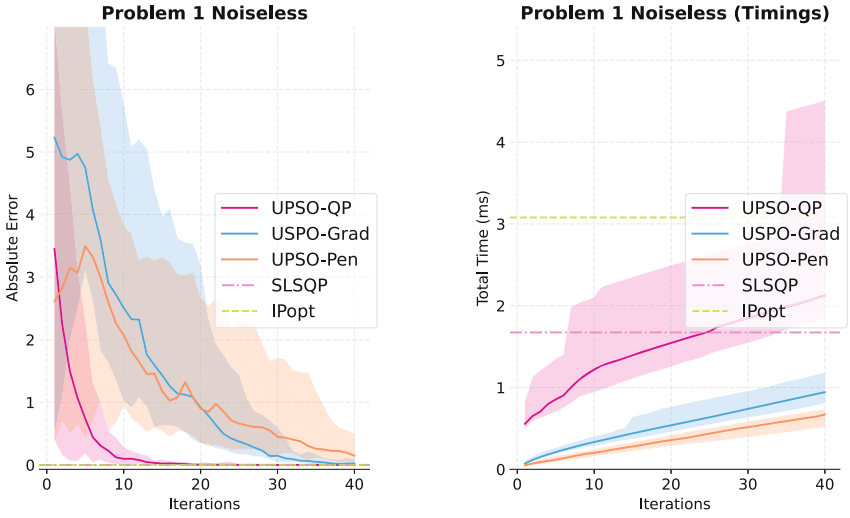

Fig. 1. Results for Problem 1. Solid lines are the median over 20 replicates and the shaded regions are the regions between the 5-th and 95-th percentiles.

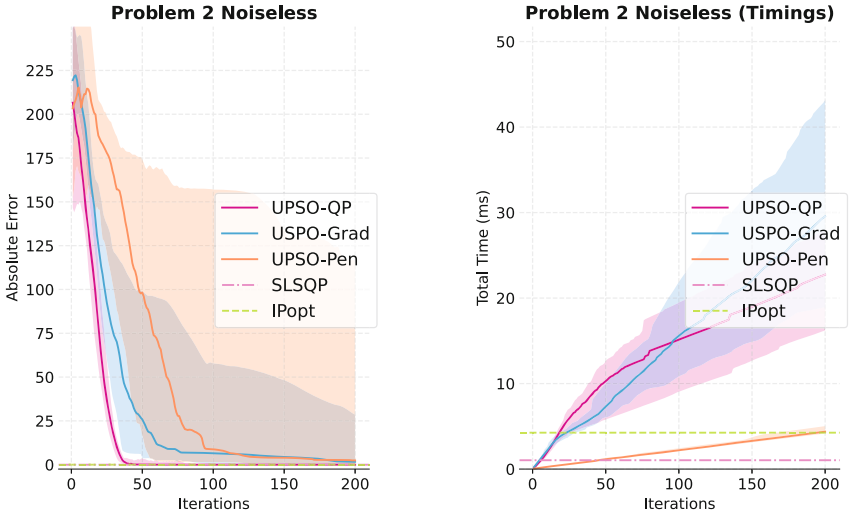

Fig. 2. Results for Problem 2. Solid lines are the median over 20 replicates and the shaded regions are the regions between the 5-th and 95-th percentiles.

Impact of Noise. The impact of imprecise information with respect to the values of the objective or constraint function can be studied and analyzed by simulating the imprecise values using, for instance, the following approach [20, 22]. Information about the function values is obtained in the form of $f^{\eta}(\boldsymbol{x})$ which determines an approximation to the true function value of the objective

function $f(\boldsymbol{x})$, contaminated by a small amount of noise η. To this end, the function values are obtained, for the case of the additive noise, as follows [8, p.40]: $f^{\eta}(\boldsymbol{x}) = f(\boldsymbol{x}) + \eta$. For the case of the multiplicative noise, the function values are obtained as follows: $f^{\eta}(\boldsymbol{x}) = f(\boldsymbol{x})(1+\eta)$, where η is a Gaussian noise term with zero mean and standard deviation σ, $\eta \sim \mathcal{N}\left(0, \sigma^2\right)$, that determines the noise strength.

Experiments. In order to showcase the effectiveness of UPSO-QP, we run each algorithm/problem pair with different noise settings: 3 noise levels for additive noise and 3 noise levels for multiplicative noise. For each problem, we select different levels in order for the noise to have an effect in performance. We also inject noise in both the objective functions and all the constraint functions. For each distinct scenario we run 20 replicates with different initial conditions. The results showcase that UPSO-QP is clearly outperforming all the other evolutionary algorithms and SLSQP (*cf.* Fig. 3, 4, 5, 6). Moreover, UPSO-QP performs as par with Ipopt in most cases and outperforms it in scenarios with big noise. We used the same hyper-parameters as in the previous section.

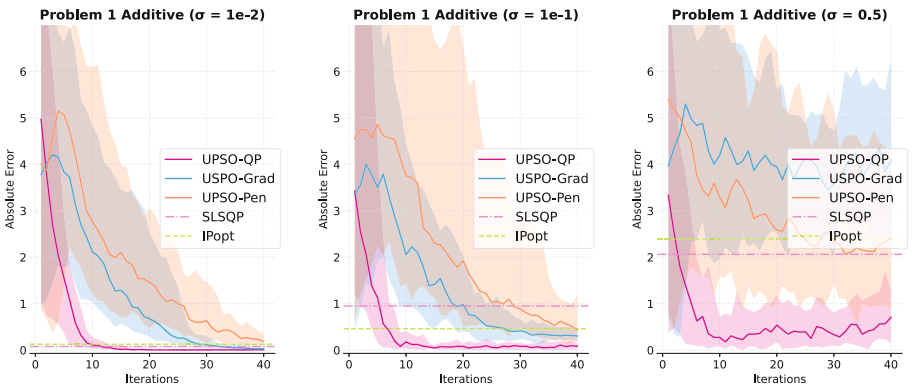

Fig. 3. Problem 1 with additive noise. Solid lines are the median over 20 replicates and the shaded regions are the regions between the 5-th and 95-th percentiles.

4.3 Evaluation on High Dimensional Problems

In this section, we will highlight the effectiveness of UPSO-QP in high dimensional and realistic examples. In particular, we will use two examples of the *Trajectory Optimization* (or Optimal Control) problem (TO) [10,16,26,28]. This type of problems tend to be quite high dimensional while also having many constraints and being sensitive to good initialization. The simplest formulation of TO problems is defined as:

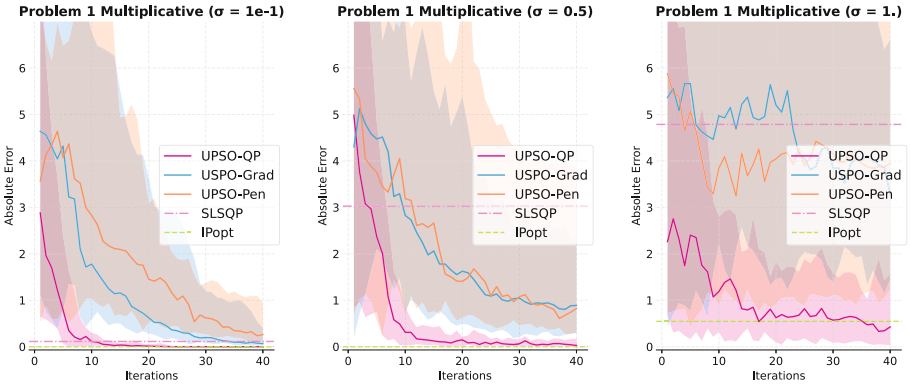

Fig. 4. Problem 1 with multiplicative noise. Solid lines are the median over 20 replicates and the shaded regions are the regions between the 5-th and 95-th percentiles.

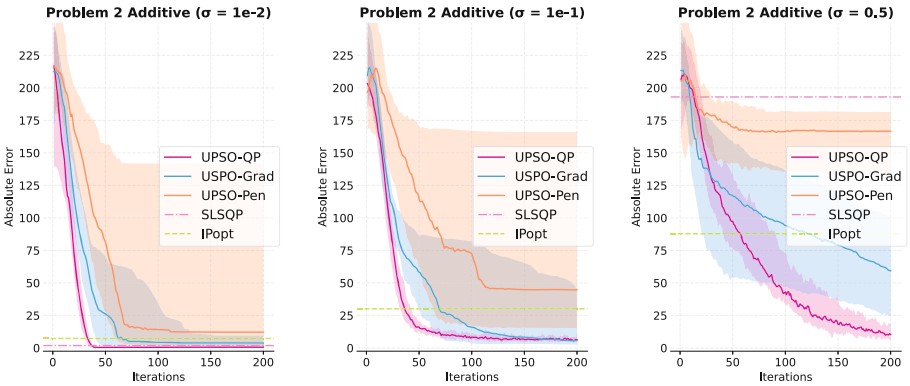

Fig. 5. Problem 2 with additive noise. Solid lines are the median over 20 replicates and the shaded regions are the regions between the 5-th and 95-th percentiles.

$$\underset{\boldsymbol{s}_1,\ldots,\boldsymbol{s}_L,\boldsymbol{u}_1,\ldots,,\boldsymbol{u}_{L-1}}{\operatorname{argmin}} \sum_{l}^{L-1} C(\boldsymbol{s}_l, \boldsymbol{u}_l) + C_{\text{final}}(\boldsymbol{s}_L),$$
$$\text{s.t.} \quad \text{Dyn}(\boldsymbol{s}_l, \boldsymbol{u}_l, \boldsymbol{s}_{l+1}) = \mathbf{0}, \tag{11}$$

where $\boldsymbol{s}_l$ is the state at time l, $\boldsymbol{u}_l$ is the action taken at time l, $C(\cdot,\cdot)$, $C_{\text{final}}(\cdot)$ define the cost functions, while $\text{Dyn}(\cdot)$ defines the dynamics equations of the system. We can additionally add more constraints depending on the problem (*e.g.* bounds on the variables). In essence, the above formulation assumes that we discretize the continuous signal at L points and enforcement all the constraints only at those points. More advanced formulations assume piece-wise polynomials and enforce the constraints on arbitrary points [28,29]. Overall, the optimization searches for the states and actions that respect the dynamics equations and minimize the costs.

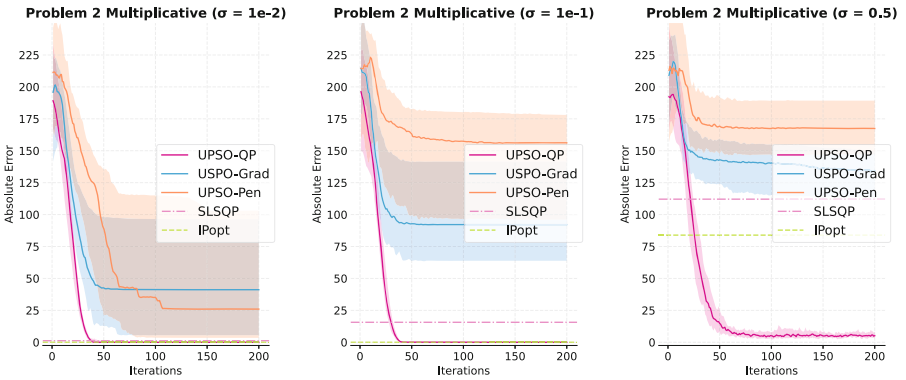

Fig. 6. Problem 2 with multiplicative noise. Solid lines are the median over 20 replicates and the shaded regions are the regions between the 5-th and 95-th percentiles.

Double Integrator. The first example that we will use is the *Double Integrator* (DI) system [23]. The DI's state is defined as $\boldsymbol{s} = \{x, \dot{x}\} \in \mathbb{R}^2$, while the actions are defined as $\boldsymbol{u} = \{\ddot{x}\} \in \mathbb{R}$. The dynamics equations of motion are given by:

$$\boldsymbol{s}_{l+1} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \boldsymbol{s}_l + \begin{bmatrix} \frac{1}{2}dt^2 \\ dt \end{bmatrix} \boldsymbol{u}_l, \tag{12}$$

where dt is the time-step of integration in seconds. In this particular setup, the system starts at $\boldsymbol{s}_1 = \{1, 0\}$ and has to reach at $\boldsymbol{s}_L = \{0, 0\}$ while minimizing the magnitude of the actions taken. $C_{\text{final}}(\boldsymbol{s}_l) = \frac{1}{2}\boldsymbol{s}_l^\top \boldsymbol{s}_l$, and $C(\boldsymbol{s}_l, \boldsymbol{u}_l) = C_{\text{final}}(\boldsymbol{s}_l) + \frac{1}{2}\,0.1\,\boldsymbol{u}_l^\top \boldsymbol{u}_l$. We use $dt = 0.1$ and $L = 51$ steps. *The total dimensions of the optimization variables is 151 (i.e.* $\boldsymbol{x} \in \mathbb{R}^{151}$*), while the total number of equality constraints are 102 (i.e.* $\boldsymbol{h}(\cdot) \in \mathbb{R}^{102}$*).* We will use this example to compare UPSO-QP to other evolutionary methods. The results showcase that UPSO-QP is able to solve the problem and converge rapidly to the optimal solution, while the other evolutionary baselines struggle at finding a good solution (*cf.* Fig. 7). This is mainly because of the dimensionality of problem and we would need to perform an extensive hyper-parameter search to make them competitive. On the contrary, UPSO-QP is able to take advantage of the local linearizations in the search space and converge quickly to the optimal value. We used the same hyper-parameters as in the previous section.

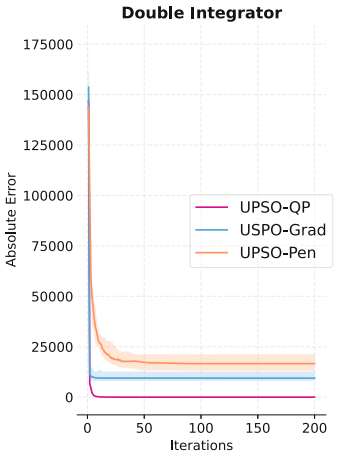

Fig. 7. Double Integrator Results. Solid lines are the median over 20 replicates and the shaded regions are the regions between the 5-th and 95-th percentiles.

Monopod Locomotion. Here we take one example of TO for legged locomotion, where the task is to generate an effective gait for a monopod robot walking on flat terrain (*cf.* Fig. 8). We follow the formulation of Winkler *et al.* [29] and:

a) Model the robot as a single rigid body mass with a leg that its mass is negligible;
b) Adopt Winkler *et al.* [29] phase-based formulation for contact switching;
c) Parameterize the body pose, foot positions and foot forces with multiple Hermite cubic polynomials.

Overall, we have the following optimization problem (omitting the cubic polynomials for clarity):

$$
\begin{aligned}
\text{find} \quad & \boldsymbol{r}(t),\ \boldsymbol{r} : \mathbb{R} \to \mathbb{R}^3, && \text{(Body positions)} \\
& \boldsymbol{\theta}(t),\ \theta : \mathbb{R} \to \mathbb{R}^3, && \text{(Body Euler angles)} \\
& \boldsymbol{p}(t),\ \boldsymbol{p} : \mathbb{R} \to \mathbb{R}^3, && \text{(Foot position)} \\
& \boldsymbol{f}(t),\ \boldsymbol{f} : \mathbb{R} \to \mathbb{R}^3, && \text{(Foot force)} \\
\text{s.t.} \quad & \text{srbd}(\boldsymbol{r}, \boldsymbol{\theta}, \boldsymbol{p}, \boldsymbol{f}) = \{\ddot{\boldsymbol{r}}, \ddot{\boldsymbol{\theta}}\}, && \text{(Dynamics)} \\
& \{\boldsymbol{r}(0), \boldsymbol{\theta}(0)\} = \{\boldsymbol{r}_{\text{init}}, \boldsymbol{\theta}_{\text{init}}\}, && \text{(Initial State)} \\
& \{\boldsymbol{r}(T), \boldsymbol{\theta}(T)\} = \{\boldsymbol{r}_{\text{goal}}, \boldsymbol{\theta}_{\text{goal}}\}, && \text{(Goal State)} \\
& \boldsymbol{p}(t) \in \mathcal{B}\big(\boldsymbol{r}(t), \boldsymbol{\theta}(t)\big), && \text{(Bounds wrt body)} \\
& \dot{\boldsymbol{p}}(t) = \boldsymbol{0},\ \text{for } t \in \text{Contact}, && \text{(No slip)} \\
& \boldsymbol{p}(t) \in \mathcal{T},\ \text{for } t \in \text{Contact}, && \text{(Contact on terrain)} \\
& \boldsymbol{f}(t) \in \mathcal{F},\ \text{for } t \in \text{Contact}, && \text{(Pushing force/friction cone)} \\
& \boldsymbol{f}(t) = \boldsymbol{0},\ \text{for } t \notin \text{Contact}, && \text{(No force in air)}
\end{aligned}
\tag{13}
$$

In this particular setup, the monopod starts at pose $\boldsymbol{r}(0) = \{0, 0, 0.5\}$, $\boldsymbol{\theta}(0) = \boldsymbol{0}$, and has to reach $\boldsymbol{r}(T) = \{1, 0, 0.5\}$, $\boldsymbol{\theta}(T) = \boldsymbol{0}$ in $T = 2\,s$, while it is allowed for 3 swing phases (foot in the air). *The total dimensions of the optimization variables is 339 (i.e.* $\boldsymbol{x} \in \mathbb{R}^{339}$*). The total number of equality constraints are 291 (i.e.* $\boldsymbol{h}(\cdot) \in \mathbb{R}^{291}$*), and the total number of inequality constraints are 225 (i.e.* $\boldsymbol{g}(\cdot) \in \mathbb{R}^{225}$*).* We will use this example to compare against Ipopt and evaluate whether UPSO-QP can be more robust to the initial solution guess. Here for UPSO-QP we used $M = 400$ particles, with 20 neighborhoods, $\chi = 0.729, c_1 = c_2 = 2.05$, $u = 0.5$, $r_{\text{qp}} = 0.005$ and $\alpha_{\text{qp}} = 1$.

The problem we are trying to solve in this section is highly non-linear, non-convex with many "bad" local optima that the optimization can be trapped around and not able to get away. For this reason and in order to test the robustness of the algorithms to the initial solution guess, we take a well-shaped initialization and add to each variable Gaussian noise $\eta \sim \mathcal{N}\left(0, \sigma^2\right)$. We vary σ from 0 to 1. This way we can have a meaningful comparison, while also getting reasonable convergence. We ran 10 replicates per experiment with different initialization parameters. The results showcase that both algorithms are able to find

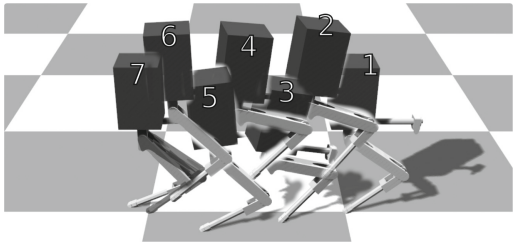

Fig. 8. Monopod: an example solution using UPSO-QP. The shaded "ghost" robot is the target. The visualizations (1–7) are snapshots at time intervals. A video of the optimized behavior is available at https://youtu.be/ZnDs8wc96eM.

the optimal solution (*cf.* Fig. 8) 100% of the time up to perturbation of $\sigma = 0.7$. For $\sigma = 0.8$ and $\sigma = 0.9$, UPSO-QP is always able to find the optimal solution, while Ipopt struggles and does not find the solution even after 5000 iterations. For $\sigma = 1$, UPSO-QP rarely (1/10 runs) finds the optimal solution before 2000 iterations. The results verify that UPSO-QP keeps the effectiveness of numerical optimization methods, while being more robust to bad initialization.

5 Concluding Remarks

We have proposed UPSO-QP, a novel algorithm that effectively combines the evolutionary and numerical optimization literature, and solves general COPs. UPSO-QP is able to keep convergence rates/wall-time similar to the analytical methods, while being robust to noisy measurements and bad initialization similar to EAs. Overall, UPSO-QP is getting the "best of both of worlds". There needs to be more investigation in which problems/scenarios the effect of the linearization part of Eq. (8) is dominant, and in which ones the PSO part dominates.

References

1. Bambade, A., et al.: PROX-QP: yet another quadratic programming solver for robotics and beyond. In: RSS 2022-Robotics: Science and Systems (2022)
2. Beck, A., Hallak, N.: The regularized feasible directions method for nonconvex optimization. Oper. Res. Lett. **50**(5), 517–523 (2022)
3. Cantú, V.H., et al.: Constraint-handling techniques within differential evolution for solving process engineering problems. Appl. Soft Comput. **108**, 107442 (2021)
4. Chatzilygeroudis, K., Cully, A., Vassiliades, V., Mouret, J.-B.: Quality-diversity optimization: a novel branch of stochastic optimization. In: Pardalos, P.M., Rasskazova, V., Vrahatis, M.N. (eds.) Black Box Optimization, Machine Learning, and No-Free Lunch Theorems. SOIA, vol. 170, pp. 109–135. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-66515-9_4
5. Chootinan, P., et al.: Constraint handling in genetic algorithms using a gradient-based repair method. Comput. Oper. Res. **33**(8), 2263–2281 (2006)

6. D'Angelo, G., Palmieri, F.: GGA: a modified genetic algorithm with gradient-based local search for solving constrained optimization problems. Inf. Sci. **547**, 136–162 (2021)
7. Elsayed, S.M., Sarker, R.A., Mezura-Montes, E.: Particle swarm optimizer for constrained optimization. In: 2013 IEEE Congress on Evolutionary Computation, pp. 2703–2711. IEEE (2013)
8. Elster, C., Neumaier, A.: A method of trust region type for minimizing noisy functions. Computing **58**, 31–46 (1997)
9. Floudas, C.A., Pardalos, P.M.: A Collection of Test Problems for Constrained Global Optimization Algorithms, vol. 455. Springer, Heidelberg (1990). https://doi.org/10.1007/3-540-53032-0
10. Hargraves, C.R., Paris, S.W.: Direct trajectory optimization using nonlinear programming and collocation. J. Guid. Control. Dyn. **10**(4), 338–342 (1987)
11. Himmelblau, D.M., et al.: Applied Nonlinear Programming. McGraw-Hill, New York (2018)
12. Jain, H., Deb, K.: An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, part ii: handling constraints and extending to an adaptive approach. IEEE Trans. Evol. Comput. **18**(4), 602–622 (2013)
13. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments - a survey. IEEE Trans. Evol. Comput. **9**, 303–317 (2005)
14. Kraft, D.: A software package for sequential quadratic programming. German Research and Testing Institute for Aerospace (1988)
15. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd edn. Springer, Heidelberg (1996). https://doi.org/10.1007/978-3-662-03315-9
16. Murray, D., Yakowitz, S.: Differential dynamic programming and newton's method for discrete optimal control problems. J. Optim. Theory Appl. **43**(3), 395–414 (1984)
17. Nocedal, J., Wright, S.: Numerical Optimization, 2nd edn., pp. 497–528. Springer, New York (2006). https://doi.org/10.1007/978-0-387-40065-5
18. Norkin, V.I.: Generalized gradients in dynamic optimization, optimal control, and machine learning problems. Cybern. Syst. Anal. **56**(2), 243–258 (2020). https://doi.org/10.1007/s10559-020-00240-x
19. Parsopoulos, K.E., Vrahatis, M.N.: Particle swarm optimization method for constrained optimization problems. In: Intelligent technologies-theory and application: New trends in intelligent technologies, vol. 76, pp. 214–220. IOS Press (2002)
20. Parsopoulos, K.E., Vrahatis, M.N.: Recent approaches to global optimization problems through particle swarm optimization. Nat. Comput. **1**, 235–306 (2002)
21. Parsopoulos, K.E., Vrahatis, M.N.: Unified particle swarm optimization for solving constrained engineering optimization problems. In: Wang, L., Chen, K., Ong, Y.S. (eds.) ICNC 2005. LNCS, vol. 3612, pp. 582–591. Springer, Heidelberg (2005). https://doi.org/10.1007/11539902_71
22. Parsopoulos, K.E., Vrahatis, M.N.: Particle Swarm Optimization and Intelligence: Advances and Applications. Information Science Publishing (2010)
23. Rao, V.G., Bernstein, D.S.: Naive control of the double integrator. IEEE Control Syst. Mag. **21**(5), 86–97 (2001)
24. Sun, Y., et al.: A particle swarm optimization algorithm based on an improved deb criterion for constrained optimization problems. PeerJ Comput. Sci. **8**, e1178 (2022)
25. Tsutsui, S., Ghosh, A.: Genetic algorithms with a robust solution searching scheme. IEEE Trans. Evol. Comput. **1**, 201–208 (1997)

26. Von Stryk, O., Bulirsch, R.: Direct and indirect methods for trajectory optimization. Ann. Oper. Res. **37**(1), 357–373 (1992)
27. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. **106**(1), 25–57 (2006)
28. Wensing, P.M., et al.: Optimization-based control for dynamic legged robots. arXiv:2211.11644 (2022)
29. Winkler, A.W., Bellicoso, C.D., Hutter, M., Buchli, J.: Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. IEEE Robot. Autom. Lett. **3**(3), 1560–1567 (2018)