CORE

Metadata, citation and similar papers at core.ac.uk

Provided by MUCC (Crossref)

J Glob Optim (2018) 71:891–913
https://doi.org/10.1007/s10898-018-0611-8

Optimal deterministic algorithm generation

Alexander Mitsos[1] · **Jaromił Najman**[1] ·
Ioannis G. Kevrekidis[2,3,4]

Received: 31 December 2016 / Accepted: 22 January 2018 / Published online: 13 February 2018
© The Author(s) 2018. This article is an open access publication

Abstract A formulation for the automated generation of algorithms via mathematical programming (optimization) is proposed. The formulation is based on the concept of optimizing within a parameterized family of algorithms, or equivalently a family of functions describing the algorithmic steps. The optimization variables are the parameters—within this family of algorithms—that encode algorithm design: the computational steps of which the selected algorithms consist. The objective function of the optimization problem encodes the merit function of the algorithm, e.g., the computational cost (possibly also including a cost component for memory requirements) of the algorithm execution. The constraints of the optimization problem ensure convergence of the algorithm, i.e., solution of the problem at hand. The formulation is described prototypically for algorithms used in solving nonlinear equations and in performing unconstrained optimization; the parametrized algorithm family considered is that of monomials in function and derivative evaluation (including negative powers). A prototype implementation in GAMS is provided along with illustrative results demonstrating cases for which well-known algorithms are shown to be optimal. The formulation is a

This project has received funding from the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) "Improved McCormick Relaxations for the efficient Global Optimization in the Space of Degrees of Freedom" MA 1188/34-1. The work of IGK was partially supported by the US National Science Foundation (CDS&E), the US AFOSR and DARPA.

✉ Ioannis G. Kevrekidis
yannis@princeton.edu

Alexander Mitsos
amitsos@alum.mit.edu

[1] AVT Process Systems Engineering (SVT), RWTH Aachen University, Forckenbeckstrasse 51, 52074 Aachen, Germany

[2] Department of Chemical and Biological Engineering and Program in Applied and Computational Mathematics, Princeton University, A319 Engineering Quad, Princeton, NJ 08544, USA

[3] IAS-TUM, Garching and Zuse Institut, FU Berlin, Berlin, Germany

[4] Present Address: Departments of Chemical and Biomolecular Engineering, Applied Mathematics and Statistics, and Urology, Johns Hopkins University, Baltimore, MD 21218, USA

mixed-integer nonlinear program. To overcome the multimodality arising from nonconvexity in the optimization problem, a combination of brute force and general-purpose deterministic global algorithms is employed to guarantee the optimality of the algorithm devised. We then discuss several directions towards which this methodology can be extended, their scope and limitations.

Keywords Optimization · Nonlinear equations · Algorithms · Optimal control

1 Introduction

Computation has led to major advances in science and engineering, in large part due to ingenious numerical algorithms. The development of algorithms is thus of crucial importance and requires substantial resources and time. Typically multiple algorithms exist for a given task. Comparisons of algorithms that tackle a given family of problems is sometimes performed theoretically and sometimes numerically. It would be desirable to automatically generate algorithms and have a guarantee that these are the best for a given problem or for a class of problems. We propose the use of numerical optimization to design *optimal algorithms*, i.e., algorithms that perform better than any conceivable alternative. More precisely, each iteration of the algorithms is interpreted, in an input-output sense, as a function evaluation. Then, an optimization problem is formulated that finds the best (in a given metric) algorithm among a family of algorithms/functions.

There is large literature on automatic generation of algorithms. In many cases genetic algorithms are used to first generate a set of algorithms built of elementary components of promising approaches. The set is evaluated on test problems and a new elitist algorithm is obtained by combination [4,21,22]. The work by [21,22] covers basic methods and ideas in the field of genetic programming. Automatic algorithm generation for retrieval, classification and other data manipulation has been proposed by [23] based on statistical data analysis and other methods. For instance, in [33] algorithms for the well-known knapsack problem are automatically generated by genetic algorithms. Algorithms are first generated for small knapsack instances and then compared to algorithms computed on larger test sets. In [19] automatic generation of parallel sorting algorithms is presented by combination of known sorting algorithms to obtain a best performing algorithm for a certain input. The well known art gallery problem (AGP) is discussed by [41], where an iterative algorithm is developed. Building on the work of [32], in [15] a novel approach for performance analysis is presented. In [20] Nesterov-like first-order algorithms for smooth unconstrained convex optimization are developed. The worst-case convergence bound of the presented algorithms is twice as small as that of Nesterov's methods. Moreover, there is substantial work on automatic code and algorithm generation for certain problems, e.g., [2,3,13,34]. Many depend on algebraic operations regarding matrices, e.g., QR decomposition, diagonal transformations or eigenvalue computation. Many of those operations are well documented in the Thesis of [6]. Finally, the problem of tuning the parameters of existing algorithms through optimization has been considered [25]. Lessard et al. [24] use control theory to derive bounds on the convergence rate of important algorithms, most of which are considered herein as well. Deterministic global optimization has been used by Ruuth and Spiteri [35,36] to maximize the Courant–Friedrichs–Lewy coefficients of Runge–Kutta methods subject to constraints ensuring that they preserve strong stability.

Herein, the focus is on iterative algorithms and the basis for the automatic generation will be a formulation similar to the ones used in optimization with dynamic systems embedded.

Consider an iterative algorithm, like the time-honored Newton–Raphson for finding roots of algebraic equations, e.g., in one dimension $x_{n+1} = x_n - f(x_n)/f'(x_n) \equiv x_n + u_n(x_n)$. The iterates can thus be seen as a "control action" u_n that depends only on the current guess, i.e., "state-dependent feedback"; a similar interpretation can be given to most algorithms. But if algorithms are feedback laws (towards a prescribed objective), then we do have celebrated mathematical tools for computing optimal feedback, e.g., Hamilton–Jacobi–Bellman. It would thus make sense to use these tools to devise optimal algorithms. Recall that iterative algorithms have been considered before in automatic algorithm generation, e.g., [41].

The key goal of the manuscript is to formulate a systematic process for obtaining optimal algorithms. To the best of our knowledge, there exists no other proposal in the literature to use deterministic global optimization for finding an algorithm that is optimal w.r.t. an objective such as the computational cost for general classes of algorithms. Recall the references [35,36] which also use deterministic global optimization; in that sense the work of Ruth and Spiteri preceeds our work in using mixed-integer nonlinear programing (MINLP) to devise optimal algorithms. However, therein the objective are maximal values of coefficients subject to stability of the algorithm, whereas we consider more general classes of algorithms and as objective estimates of the cost of the algorithm. More specifically, in our work, optimality is determined based on a desired measurable performance property which is encoded as the objective of the optimization problem. For instance this can be the computational expense of the algorithm, possibly accounting for the memory requirements with an appropriate weight. In the article we use an approximation of the cost, and throughout determine optimality and cheapest algorithms in the sense of minimizing this approximated cost. Upon convergence of our formulation, the resulting algorithm is optimal among a considered family of algorithms, which in turn is encoded using the variables (or degrees of freedom) of the optimization problem. The optimization formulation is completed by the constraints which ensure that only feasible algorithms will be selected among the family of algorithms considered, i.e., algorithms that converge to a solution of the problem at hand. The formulations can be cast as nonlinear programs (NLP) or MINLP. These formulations are quite expensive; in fact, as we discuss in the following they are expected to be more expensive than the original problem; however, we argue that this is acceptable.

Our proof of concept illustrations involve devising optimal algorithms that perform local unconstrained minimization of scalar functions and those that solve systems of equations. We consider a class of algorithms that involves evaluations of the function and its first two derivatives. Algorithms can be composed for a modest number of iterations of a monomial involving these quantities and we call these monomial-type algorithms. Optimality is first sought within this class of algorithms. The class is then extended to allow for methods such as Nesterov's acceleration. The well-known steepest descent is shown to be optimal in finding a stationary point of a univariate fourth-order polynomial, whereas it is infeasible for the two-dimensional Rosenbrock objective function (it would take more than the allowed number of iterations to converge). Both statements hold for a given initial guess (starting value of the variables given to the algorithm). We also consider the effect of different initial guesses and show that Nesterov's method is the cheapest in an ensemble of initial guesses. Finally, we focus on finding optimal algorithms for a given problem, e.g., minimizing a fixed function; however, we discuss how to handle multiple instances by considering families of problems.

The remainder of the manuscript is structured starting with the simplest possible setting, namely solution of a single nonlinear equation by a particular class of algorithms. Two immediate extensions are considered (a) to equations systems and (b) to local unconstrained optimization. Subsequently, numerical results are presented along with some discussion.

Limitations are discussed followed by potential extensions to more general problems and algorithm classes. Finally, key conclusions are presented.

2 Definitions and assumptions

For the sake of simplicity, first, solution of a single nonlinear equation is considered as the prototypical task an algorithm has to tackle.

Definition 1 (*Solution of equation*) Let $x \in X \subset \mathbb{R}$ and $f : X \to \mathbb{R}$. A solution of the equation is a point $x^* \in X$ with $f(x^*) = 0$. An approximate solution of the equation is a point $x^* \in X$ with $|f(x^*)| \leq \varepsilon$, $\varepsilon > 0$.

Note that no assumptions on existence or uniqueness are made, nor convexity of the function. A particular interest is to find a steady-state solution of a dynamic system $\dot{x}(t) = f(x(t))$. We will first make use of relatively simple algorithms:

Definition 2 (*Simple Algorithm*) Algorithms operate on the variable space $\mathbf{x} \in X \subset \mathbb{R}^{n_x}$ and start with an initial guess $\mathbf{x}^{(0)} \in X$. A given iteration (it) of a simple algorithm amounts to calculating the current iterate as a function of the previous $\mathbf{x}^{(it)} = g_{it}\left(\mathbf{x}^{(it-1)}\right)$ with $g_{it} : X \to X$. Therefore, the algorithm has calculated $\mathbf{x}^{(it)} = g_{it}\left(g_{it-1}\left(\dots(g_1(\mathbf{x}^{(0)}))\right)\right)$ after iteration (it). A special case are algorithms that satisfy $g_{it_1}(\mathbf{x}) = g_{it_2}(\mathbf{x})$ for all it_1, it_2, i.e., use the same function at each iteration.

Note that the definition includes the initial guess $\mathbf{x}^{(0)}$. Herein, both a fixed initial guess and an ensemble of initial conditions is used. Unless otherwise noted, the same function will be used at each iteration $g_{it} \equiv g$.

In other words, algorithmic iterations can be seen as functions and algorithms as composite functions. This motivates our premise, that finding an optimal algorithm constitutes an optimal control problem. In some sense, the formulation finds the optimal function among a set of algorithms considered. One could thus talk of a "hyper-algorithm" or "meta-algorithm" implying a (parametrized) family of algorithms, or possibly the span of a "basis set" of algorithms that are used to identify an optimal algorithm. In the following we will assume $X = \mathbb{R}^{n_x}$ and consider approximate feasible algorithms:

Definition 3 An algorithm is *feasible* if it solves a problem; otherwise it is termed *infeasible*. More precisely: it is termed *feasible in the limit* for a given problem if it solves this problem as the number of iterations approaches ∞; it is termed *finitely feasible* if it solves the problem after a finite number of iterations; it is termed *approximate feasible* if after a finite number of iterations it solves a problem approximately. In all these cases the statements hold for some (potentially open) set of initial conditions. A feasible algorithm is *optimal* with respect to a metric if among all feasible algorithms it minimizes that metric.

One could also distinguish between feasible path algorithms, i.e., those that satisfy $\mathbf{x}^{(it)} \in X$, $\forall it$ in contrast to algorithms that for intermediate iterations violate this condition.

Throughout the article, it is assumed that $f : X \subset \mathbb{R} \to \mathbb{R}$ is sufficiently smooth, in the sense that if algorithms are considered that use derivatives of a given order, then f is continuously differentiable to at least that order. The derivative of order j is denoted by $f^{(j)}$ with $f^{(0)} \equiv f$. A key point in our approach is the selection of a class of algorithms: a (parametrizable) family of functions. It is possible, at least in principle, to directly consider the functions g_{it} and optimize in the corresponding space of functions. Alternatively, one can identify a specific family of algorithms, e.g., those based on gradients, which includes well-known algorithms such as Newton and explicit Euler:

Definition 4 Consider problems that involve a single variable $x \in X \subset \mathbb{R}$ and $f : X \rightarrow \mathbb{R}$. *Monomial-type algorithms* are those that consist of a monomial $g_{it}(x) = x + \alpha_{it} \Pi_{j=0}^{j_{max}} \left(f^{(j)}(x)\right)^{\nu_j}$ of derivatives of at most order j_{max}, allowing for positive and negative (integer) powers ν_j.

As examples of algorithms in this family consider the solution of a single nonlinear equation, Definition 1, by the following important algorithms

- Newton: $x^{(it)} = x^{(it-1)} + \frac{f^{(0)}(x^{(it-1)})}{f^{(1)}(x^{(it-1)})}$, i.e., we have $\alpha = 1$, $\nu_0 = 1$, $\nu_1 = -1$, $\nu_{i>1} = 0$.
- Explicit Euler: $x^{(it)} = x^{(it-1)} + f(x^{(it-1)})\Delta t$, i.e., we have $\alpha = \Delta t$, $\nu_0 = 1$, $\nu_{i>0} = 0$.

3 Development

We first develop an optimization formulation that identifies optimal algorithms of a particular problem: the solution of a given nonlinear scalar equation, Definition 1. We will consider approximate feasible solutions and simple algorithms that use the same function at each iteration. If the optimization metric accounts for the computational cost of the algorithm, then it is expected that algorithms that are only feasible in the limit will not be optimal since their cost will be higher than those of finitely feasible algorithms.

Obviously to solve the optimization problem with local methods one needs an initial guess for the optimization variables, i.e., an initial algorithm. Some local optimization methods even require a "feasible" initial guess, i.e., an algorithm that solves (non-optimally) the problem at hand. Note however, that we will consider deterministic global methods for the solution of the optimization problem, and as such, at least in theory, there is no need for an initial guess.

3.1 Infinite problem

We are interested in devising algorithms as the solution of an optimization problem. The key idea is to include the iterates of the algorithm $x^{(it)}$ as (intermediate) *optimization variables* that are determined by the algorithmic iteration. By imposing a maximal number of iterations, we have a finite number of variables. The algorithm itself consists of optimization variables, albeit infinite (optimal control). Moreover, an end-point constraint ensures that only feasible algorithms are considered. Finally, an optimization metric is required which maps from the function space to $\mathbb{R}$, such as the computational cost. We will make the assumption that the objective is the sum of a cost at each iteration which in turn only depends on the iterate, and implicitly on the problem to be solved.

$$\begin{aligned} \min_{g, x^{(it)}, it_{con}} \ & \sum_{it=1}^{it_{con}} \phi\left(g\left(x^{(it)}\right)\right) \\ \text{s.t. } & x^{(it)} = g\left(x^{(it-1)}\right), \quad it = 1, 2, \ldots, it_{con} \\ & \left|f\left(x^{(it_{con})}\right)\right| \leq \epsilon, \end{aligned} \tag{1}$$

where it_{con} corresponds to the iteration at which the desired convergence criterion is met. The optimal point found embodies an optimal algorithm g^*; depending on the method used it may be a global, an approximately global or a local optimum. For notational simplicity, we have assumed that the minimum is attained. It is outside the scope of this manuscript to determine if the infimum is attained, e.g., if g can be described by a finite basis of a finite vector space.

The intermediate variables can be considered as part of the optimization variables along with the constraints or eliminated. Note the analogy to full discretization vs. late discretization vs. multiple shooting in dynamic optimization [5,30] with known advantages and disadvantages.

Formulation (1) is not a regular optimal control problem as the number of variables is not a priori known. There are however several straightforward approaches to overcome this problem, such as the following. A maximal number of iterations it_{max} is considered a priori and dummy variables for $it_{con} < it < it_{max}$ are used along with binary variables capturing if the convergence has been met. The cost is accounted only for the iterates before convergence:

$$\min_{g,x^{(it)}} \sum_{it=1}^{it_{max}} y_{res}^{(it)} \phi\left(g\left(x^{(it)}\right)\right)$$

$$\text{s.t. } x^{(it)} = g\left(x^{(it-1)}\right), \quad it = 1, 2, \ldots, it_{max} \tag{2}$$

$$y_{res}^{(it)} = \begin{cases} 0 & \text{if } \left|f\left(x^{(it)}\right)\right| < \varepsilon \\ 1 & \text{else} \end{cases}, \quad it = 1, 2, \ldots, it_{max} \tag{3}$$

$$y_{res}^{(it_{max})} = 0, \tag{4}$$

where satisfaction of the last constraint ensures convergence. In “Appendix B” an alternative is given. In the formulation above each iteration uses the same function g. Allowing different functional forms at each iteration step, is of interest, e.g., to devise hybrid iteration algorithms, such as few steepest descent steps followed by Newton steps.

3.2 Finite problem

We now use the family of monomial-type algorithms, Definition 4, as our ensemble of algorithms. In analogy to the infinite case we assume that the objective depends on the evaluation of derivatives (of the result of the algorithm with respect to its inputs), which in turn only depends on the iterate, and implicitly on the problem to be solved. To account for the expense of manipulating the derivatives, e.g., forming the inverse of a matrix (recall that simple algorithms scale cubically in the size, i.e., the number of rows of the matrix), we assume that the cost also involves an exponent:

$$\min_{\nu_j,\alpha_{it},x^{(it)},it_{con}} \sum_{it=1}^{it_{con}} \sum_{j=0}^{j_{max}} \phi_j\left(f^{(j)}\left(x^{(it)}\right), \alpha_{it}, \nu_j\right)$$

$$\text{s.t. } x^{(it)} = x^{(it-1)} + \alpha_{it} \Pi_{j=0}^{j_{max}} \left(f^{(j)}(x^{(it-1)})\right)^{\nu_j}, it = 1, \ldots, it_{con} \tag{5}$$

$$\left|f\left(x^{(it_{con})}\right)\right| \leq \epsilon,$$

where again the number of variables is not known a priori, see above. Note that, since the exponents are integer-valued, we have an MINLP and thus, for a given function considered (i.e., for f fixed), we expect that the minimum is attained. Again the intermediate variables can be considered as part of the optimization variables along with the constraints, or they can be eliminated. The optimal point found gives optimal step size α^* as well as exponents

ν_j^*. Herein, we will restrict the step size to discrete values, mimicking line-search methods, but in general step size can be optimized over a continuous range. Allowing for a different algorithm found at each step ($\nu_j^{it_1} \neq \nu_j^{it_2}$), the number of combinations dramatically increases but there is no *conceptual* difficulty. The form is not common for an MINLP but can be easily converted to a standard one as shown in "Appendix C".

It may not always be easy to estimate the cost $\phi_j\left(f^{(j)}\left(x^{(it)}\right), \alpha_{it}, \nu_j\right)$. For instance consider Newton's method augmented with a line-search method $x^{(it)} = x^{(it-1)} + \alpha_{it}\frac{f(x^{(it-1)})}{f^{(1)}(x^{(it-1)})}$. For each major iteration, multiple minor iterations are required, with evaluations of f. If the cost of the minor iterations is negligible, the computational cost is simply the evaluation of f, its derivative and its inversion. If we know the number of minor iterations then we can calculate the required number of evaluations. If the number of iterations is not known, we can possibly obtain it by the step size, but this requires knowledge of the line search strategy. In some sense this is an advantage of the proposed approach: the step size is not determined by a line search method but rather by the optimization formulation. However, a challenge is that we need to use a good cost for the step size, i.e., incorporate in the objective function the computational expense associated with the algorithm selecting a favorable step size α_{it}. Otherwise, the optimizer can select an optimal α_{it} for that instance which is not necessarily sensible when the cost is accounted for. To mimic the way conventional algorithms work, in the numerical results we allow discrete values of the step size $\alpha_{(it)} = \pm 2^{-\bar{\alpha}_{it}}$ and put a bigger cost for bigger $\bar{\alpha}$ since this corresponds to more evaluations of the line search.

The algorithm is fully determined by selecting ν_j and $\bar{\alpha}_{it}$. So in some sense the MINLP is a purely integer problem: the variables $x^{(it)}$ are uniquely determined by the equations. To give a sense of the problem complexity, consider the setup used in the numerical case studies. Assume we allow derivatives of order $j \in \{0, 1, 2\}$ and exponents $\nu_j \in \{-2, -1, 0, 1, 2\}$ and these are fixed for each iteration it. This gives $5^3 = 125$ basic algorithms. Further we decide for each algorithm and iteration it if the step size α_{it} is positive or negative, and allow $\bar{\alpha}_{it} \in \{0, 1, \dots, 10\}$, different for each iteration it of the algorithm. Thus, we have 2×10^5 combinations of step sizes for each algorithm and 25×10^6 total number of combinations.

3.3 System of equations

A natural extension of solving a single equation is to solve a system of equations.

Definition 5 Let $\mathbf{x} \in X \subset \mathbb{R}^{n_x}$ and $\mathbf{f} : X \to \mathbb{R}^{n_x}$. A solution of the system of equations is a point $\mathbf{x}^* \in X$ with $\mathbf{f}(\mathbf{x}^*) = \mathbf{0}$. An approximate solution of the system of equations is a point $\mathbf{x}^* \in X$ with $||\mathbf{f}(\mathbf{x}^*)|| \leq \varepsilon, \varepsilon > 0$.

In addition to more cumbersome notation, the optimization problems become much more challenging to solve due to increasing number of variables and also due to the more expensive operations. Obviously the monomials need to be defined appropriately; for instance the inverse of the first derivative corresponds to the inverse of the Jacobian, assuming it has a unique solution. This in turn can be written as an implicit function or a system of equations can be used in the optimization formulation. For the numerical results herein we use small dimensions (up to two herein) and analytic expressions for the inverse. In "Appendix A" we discuss an approach more amenable to higher dimensions.

3.4 Special two-step methods

A particularly interesting class of algorithms are the so-called Nesterov's methods, see, e.g., [39]. These compute an intermediate iterate $y^{(it)} = h(x^{(it)}, x^{(it-1)})$ and use it for the computation of the next value $x^{(it)}$. The above formulation does not allow such intermediate calculations but can be relatively easily extended to

$$\begin{aligned}
\min_{g,x^{(it)},it_{con}} \quad & \sum_{it=1}^{it_{con}} \phi\left(g\left(x^{(it)}\right)\right) \\
\text{s.t.} \quad & x^{(it)} = g\left(y^{(it-1)}\right), \quad it = 1, 2, \ldots, it_{con} \\
& y^{(it)} = x^{(it)} + \beta_{it}\left(x^{(it)} - x^{(it-1)}\right), \quad it = 1, 2, \ldots, it_{con} \\
& \left|f\left(x^{(it_{con})}\right)\right| \le \epsilon.
\end{aligned} \tag{6}$$

This latter formulation covers most of Nesterov's constant schemes along with potentially new schemes. The introduction of the parameter β necessitates capturing the cost for different values of β. Herein, we add the value of β to the term of each iteration for the objective function. We note that this choice is somewhat questionable but for the illustrative, proof-of-concept computations herein, is acceptable.

3.5 Finding optimal local optimization algorithms

It is relatively easy to adapt the formulation from equation solving to local solution of optimization problems. The only substantial change consists in replacing the end-point constraint with some desired termination criterion such as an approximate solution of the KKT conditions. For unconstrained problems the corresponding criterion is stationarity of the objective $\frac{\partial f}{\partial x_{ix}} = 0$, for $ix = 1, \ldots, n_x$. Similarly to systems of equations, increased dimensionality ($n_x > 1$) implies that vectors and matrices arise and operations need to be appropriately formulated.

4 Numerical results

4.1 Method

We provide a prototype implementation in GAMS [11], inspired by [29]. We used GAMS 24.5 and tested both local and global methods, mostly KNITRO [12] and BARON [40]. All the common elements are placed in "`optforalgomain.gms`". The problem-specific definitions are then given in corresponding GAMS files that are included so it is easy to switch between problems. To limit the number of variables we avoid introducing variables for f and its derivatives and rather define them using macros. We do however introduce auxiliary variables for the iterates of the algorithm as well as variables that capture the residuals, see below. Both could be avoided by the use of further macros. Since we utilize global solvers we impose finite bounds for all variables.

As discussed, we encode the choice of derivative as a set of binary variables y_j^k.

$$\sum_{k=k_{min}}^{k_{max}} y_j^k \left(f^{(j)}(x^{(i-1)})\right)^k \text{ with } \sum_{k=k_{min}}^{k_{max}} y_j^k = 1.$$

Similarly we encode the step size using discrete values $\pm 1/2^{\alpha_{it}}$. Two problems are considered, namely solution of systems of equations and minimization, and in both cases we consider one-dimensional and two-dimensional problems. The minimization is assumed unconstrained, but since we impose bounds on the variables we essentially assume knowledge of an open box in which optimal solutions are found. We allow 10 steps of the algorithm. We require that at least the last iterate meets the desired convergence criteria.

For each iterate it we calculate the residuals by introducing a real-valued variable $res^{(it)}$ and a binary $y_{res}^{(it)}$ denoting if the desired convergence is met ($y_{res}^{(it)} = 0$) or not ($y_{res}^{(it)} = 1$). For the case of equation solving we take as residuals the value of the right-hand-side of the equation (function) $res^{(it)} = \max_{if} |f_{if}\left(\mathbf{x}^{(it)}\right)|$ and in the case of minimization the value of the derivative $res^{(it)} = \max_{ix} \left| \left(\frac{\partial f}{\partial x_{ix}}\right)\Big|_{\mathbf{x}^{(it)}} \right|$. This corresponds to the infinity norm, which is chosen since initial testing suggests more robust optimization. The absolute values are written as pair of inequalities, e.g., for equation solving

$$res^{(it)} \geq \pm f_{if}\left(\mathbf{x}^{(it)}\right) \forall if.$$

Then the binary variable is decided based on a standard big-M constraint

$$M_{res} y_{res}^{(it)} \geq (res^{(it)} - \varepsilon),$$

where M_{res} is the big-M constant that captures the magnitude of the residual res. Only iterations that do not meet the residual tolerance are accounted for in the objective

$$\min \sum_{it=1}^{it_{max}} y_{res}^{(it)} \left(\bar{\alpha}_{it} + \sum_{j=1}^{j_{max}} \sum_{k=k_{min}}^{k_{max}} y_j^k \phi_j^k \right)$$

assuming nonnegative costs ϕ_j^k. In the numerical experiments, we assume for ϕ_j^k a zero cost for zero exponent, a cost of 1 for exponent 1, a cost of 1.5 for an exponent of 2, a cost of 2 for an exponent of − 1 and a cost of 3 for an exponent of − 2, i.e., the function ϕ_j^k is a discrete function that attains the predefined values. The idea is to capture the expense of inversion and exponentiation. These costs are multiplied by 1 for f, 10 for $f^{(1)}$ and 100 for $f^{(2)}$.

The problems considered are

(EQx3) Solving $x^3 = 1$, $x \in [-2, 2]$, $x^{(0)} = 0.1$
(EQexp) Solving $x \exp(x) = 1$, $x \in [-2, 2]$, $x^{(0)} = 0.1$
(minx4) $\min_x x^4 + x^3 - x^2 - 1$, $x \in [-2, 2]$, $x^{(0)} = 0.1$
(RB) Rosenbrock function in 2d $\min_{\mathbf{x}} 100(x_2 - x_1^2)^2 + (1 - x_1)^2$, $\mathbf{x} \in [-2, 2]^2$, $\mathbf{x}^{(0)} = (0.7, 0.75)$
(eqn2d) Simple 2d equation solving $x_2 - x_1^2 = 0$, $5x_2 - \exp(x_1) = 0$. Here we excluded second derivatives.
(convex) $\min_x \exp(x) + x^2$ (strongly convex with convexity parameter 2), $x \in [-2, 2]$, $x^{(0)} = 0.1$
(quad2d) $\min_{\mathbf{x}} (x_1 - 1)^2 + 2x_2^2 - x_1 x_2$, $\mathbf{x} \in [-1, 2]^2$, $\mathbf{x}^{(0)} = (-1, -1)$
(exp2d) $\min_{\mathbf{x}} \exp(x_1) + x_1^2 + \exp(x_2) + x_1^2 + x_1 x_2$, $\mathbf{x} \in [-1, 1]^2$, $\mathbf{x}^{(0)} = (0.5, 0.5)$. The optimum of f is at $\mathbf{x}^* \approx (-0.257627653, -0.257627653)$

4.2 Single-step methods

We tried several runs, with short times (order of minute) and longer times (10 min). BARON does not utilize KNITRO as a local solver. We made the experience that KNITRO performs very well in finding feasible points for hard integer problems. Therefore, we let KNITRO find a feasible point for each fixed algorithm, i.e., we fixed variables describing the algorithm to reduce the problem's dimension, to provide a good initial point for BARON. Note that when we let KNITRO search the whole algorithm space, it often fails to find any feasible algorithm, even when they exist. This again shows the hardness of the formulation. In some cases convergence of the optimization was improved when we enforced that residuals are decreasing with iteration $res^{(it)} < res^{(it-1)}$. It is noteworthy that in early computational experiments the optimizer also managed to furnish unexpected solutions that resulted in refining the formulation, e.g., choosing a step size α that resulted in direct convergence before discrete values of α were enforced.

Due to the relatively difficult convergence of the optimization and the relatively small number of distinct algorithms (125), we solved each problem for each algorithm in three sequential steps: first KNITRO to find good initial guesses, then BARON without convergence criterion (i.e., without imposing $y_{res}^{(it_{con})} = 0$) and then BARON with convergence criterion. We tried a 60 s and a 600 s run for each problem. We can see that in 600 s the optimizer was able to detect more algorithms as (in)feasible, see Table 1. Still, the formulation without a fixed algorithm could not be solved to optimality within 600 s. Convergence to the global optimum is not guaranteed for all cases. The main findings of this explicit enumeration is that most algorithms are infeasible for harder problems. For equation-solving, in addition to Newton's algorithm, some algorithms with second derivatives are feasible. In, e.g., problem (minx4) the derivatives could attain 0 leading to a numerical problem. The solvers were still able to fight this issue in most cases and cases where algorithms resulted in an undefined operation were skipped.

The algorithms discovered as optimal for unconstrained minimization are instructive (at least to us!). In the problem (minx4) the steepest descent algorithm is optimal. It is interesting to note that the algorithms do not furnish a global minimum: in the specific formulation used optimal algorithms are computationally cheap algorithms that give a stationary point. In contrast, for the minimization of the Rosenbrock function (RB), steepest descent is not feasible within ten iterations, which constitutes a well-known behavior. In the case of solving a single equation, several feasible algorithms were obtained.

Table 1 shows that with more time, the solvers are able to solve more problems and verify more algorithms as (in)feasible. For (RB), no algorithm was found feasible, since we only allowed for ten iterations while finding the minimum of the Rosenbrock function is known to be hard and needs way more than 10 iterations in general. Note that in the case of the (RB) problem, Newton's algorithm with optimal line search is theoretically feasible within 10 iterations but was not found by the solver. It is also notable that for the (exp2d) problem the number of solved problems is the same but the solver was able to identify more algorithms as feasible within 60 s than in 600 s. We found that two algorithms for (exp2d) were found infeasible in 600 s while the same instances for the algorithms were not solved in 60 s. Moreover, in 60 s two algorithms were found to be feasible while the solver was not able to verify feasibility of those in 600 s. We remind the reader that the complexity of the problems (EQx3)–(exp2d) for which we search for optimal algorithms, does not reflect the complexity of the MINLP formulations (5) and (6). In that sense, the formulations we used run into the limits of the general-purpose state-of-the-art deterministic global optimization solvers. This motivates the development of a more advantageous formulation to exploit the problem's structure as well as the development of tailored MINLP heuristics and algorithms; both extensions are outside the scope of this article.

We also used a genetic algorithm approach in order to solve problems (EQx3) and (EQexp) with the one-step formulation. We used the genetic algorithm implemented in MATLAB R2016a. The genetic algorithm could not find a feasible solution after 10^9 generation steps. Since the nonlinear equations in the formulation make the problem very hard for the GA,

Table 1 Table summarizing the numerical results for the presented problems

Problem	With CC				w/o CC	
	60 s		600 s		60 s	600 s
	#sol	#feas	#sol	#feas	#sol	#sol
(EQx3)	39	2	41	4	41	42
(EQexp)	31	25	61	54	28	59
(minx4)	13	13	15	15	15	18
(RB)	0	0	0	0	1	1
(eqn2d)	32	0	45	0	39	43
(convex)	52	52	52	52	55	56
(quad2d)	38	12	39	13	39	40
(exp2d)	53	41	53	39	54	55

We allowed at most 10 iterations. CC stands for Convergence Criterion. #sol represents the number of solved problems out of 125 and #feas stands for the number of feasible algorithms. Note that in the cases without convergence criterion, each solved problem was also automatically feasible. The problems (convex), (quad2d) and (exp2d) were solved using the two-step formulation while the rest of the problems were solved using the one-step formulation

we relaxed these manually by reformulating them into two inequalities with an additional $\epsilon = 0.001$ factor to allow for a larger feasible set. Still, the GA was not able to find a feasible solution point. We also initialized the GA with one of the feasible algorithms but it did not help either. We do not claim that a more sophisticated genetic algorithm with a possibly differently tailored formulation would be able to solve the kind of problems considered in this work, even though this result gives us a first positive impression of our approach.

4.2.1 Unexpected algorithms discovered as optimal

Two particularly interesting algorithms identified (with $\alpha_{it} < 0$) are

$$x^{(it)} = x^{(it-1)} + \alpha_{it} \frac{f^{(0)}(x^{(it-1)}) f^{(1)}(x^{(it-1)})}{f^{(2)}(x^{(it-1)})}, \tag{7}$$

and

$$x^{(it)} = x^{(it-1)} + \alpha_{it} \frac{f^{(0)}(x^{(it-1)})(f^{(1)}(x^{(it-1)}))^2}{(f^{(2)}(x^{(it-1)}))^2} \tag{8}$$

which can be seen as a combination of Newton's algorithms for equation solving and unconstrained optimization. The step sizes found vary from iteration to iteration, but the algorithms converge also with $\alpha = -1$.

There is an algorithm mentioned between the lines of section 3.1.2 in [14] for root finding of *convex* functions, that is of high interest when examining algorithm (7) because it seems to cover the basic idea behind (7). The algorithm given in [14] is:

$$x^{(it)} = x^{(it-1)} - s^{(it-1)} f^{(0)}(x^{(it-1)}) f^{(1)}(x^{(it-1)}), \; s^{(it-1)} > 0, \tag{9}$$

where $s^{(it-1)}$ denotes the step size. In [14], there is also a Lemma guaranteeing the existence of an appropriate $s^{(it-1)}$ for each iteration. There is also a discussion about a step size strategy and the convergence rate of the iterative algorithm (9) is shown to be "linear even for bad initial guesses - however, possibly arbitrarily slow" [14], i.e., with appropriately chosen $s^{(it-1)}$ in each iteration the algorithm converges but not faster than linear. Additionally, $s^{(it-1)}$ can be chosen well enough to converge to a root, but still such that the convergence of the algorithm is arbitrarily slow. In [14], it is also said that the so-called "pseudo-convergence" characterized by "small" $\| f^{(1)}(x) f(x) \|$ may occur far from the solution point due to local ill-conditioning of the Jacobian matrix.

It is relatively easy to see that algorithm (7) is a special case of (9), and to rationalize why it does work for convex functions (or for concave functions by changing the sign of α). Recall that we optimized for each possible algorithm that uses up to second derivatives. Algorithm (7) was discovered for the equation (EQexp). In this special case, step size $\frac{1}{f^{(2)}(x^{(it-1)})}$ was good enough to converge within the 10 allowed iterations and α can be set to -1. We could rethink the costs of a changing α and the usage of the inverse of the second derivative in order to force $1 \neq \alpha \neq -1$.

It is interesting to consider the convergence of such algorithms for the case that the root of the function f exists only in the limit. Then, the algorithms searching for the roots can only reach ϵ-convergence. For instance, consider $f(x) = x\exp(x)$, which is convex over, e.g., $X = (-\infty, -2]$. It holds that $\lim_{x\to-\infty} f(x) = 0$ and for, e.g., the starting point $x^{(0)} = -5$, algorithm (7) moves in the negative x-direction, i.e., the algorithm iterates to the left in each iteration. Let us now consider the algorithm given by (8). Here $\frac{f^{(1)}(x^{(it-1)})}{(f^{(2)}(x^{(it-1)}))^2}$ operates as the step size $s^{(it-1)}$. Algorithm (8) is more problematic, but in the special case considered here, it converges.

All in all, $\frac{1}{f^{(2)}(x^{(it-1)})}$ and $\frac{f^{(1)}(x^{(it-1)})}{(f^{(2)}(x^{(it-1)}))^2}$ can act as step size estimators in algorithm (9) for specific functions. Nevertheless, the value of the algorithms furnished is questionable. We show an example where $\frac{1}{f^{(2)}(x^{(it-1)})}$ and $\frac{f^{(1)}(x^{(it-1)})}{(f^{(2)}(x^{(it-1)}))^2}$ cannot act as step sizes and the algorithm would not be feasible for $\alpha = -1$. Consider the function $f(x) = x^2 - 3$ with $f^{(1)}(x) = 2x$ and $f^{(2)}(x) = 2$. Algorithms (7) and (8) both diverge for all $x \in \mathbb{R}$ except for the roots and the minimum, and some special cases where the step size $\frac{1}{f^{(2)}(x^{(it-1)})} = \frac{1}{2}$ or $\frac{f^{(1)}(x^{(it-1)})}{(f^{(2)}(x^{(it-1)}))^2} = \frac{x}{2}$ is exact, e.g., $x^{(0)} \in \{-2, 2\}$. For example, considering (7) with the starting points $x_1^{(0)} = 3$ and $x_2^{(0)} = 0.1$, we get the following iteration sequence:

$$\begin{aligned}
x_1^{(0)} &= 3, & x_2^{(0)} &= 0.1\\
x_1^{(1)} &= -15, & x_2^{(1)} &= 0.399\\
x_1^{(2)} &= 3315, & x_2^{(2)} &= 1.532478801\\
x_1^{(3)} &= -36429267622, & x_2^{(3)} &= 2.53090211\\
x_1^{(4)} &= \ldots, & x_2^{(4)} &= \ldots
\end{aligned}$$

showing divergence of algorithm (7) for this simple convex function $f(x) = x^2 - 3$. Algorithm (8) shows similar behavior. Although f is convex, the algorithms both diverge showing that the step size estimators in (7) and (8) given by $\frac{1}{f^{(2)}(x^{(it-1)})}$ and $\frac{f^{(1)}(x^{(it-1)})}{(f^{(2)}(x^{(it-1)}))^2}$ cannot be optimal for all convex functions.

4.3 The importance of the step size

Recall that the optimization formulation allows choice of the step size aiming to mimic typical step size search. In some cases this leads to "spurious" values of the step size α. We allow step sizes $\alpha_{it} = \pm 2^{-\bar{\alpha}_{it}}$ with $\bar{\alpha}_{it} \in \{0, 1, \ldots, 10\}$. The algorithms discovered by our formulation might, in some cases, be rationalized as algorithms for optimal line search. There are many rules for the choice of the step size when using line-search. One of those rules can be adjusted to only use step sizes of size $\alpha_{it} = \pm 2^{-\bar{\alpha}_{it}}$. The rule says then to divide the step size by 2 as long as the current step is not feasible, i.e., the step obtained in the end does not have to be optimal but only feasible. Herein, the optimizer finds the *optimal* step size for the line-search using this rule for the given function f and the given iterative algorithm. In other words, in the 1D case, step size allows each algorithm to be feasible. Good algorithms for the general case, e.g., Newton, will have a sensible step size selection, whereas other algorithms may have an apparently spurious one. In particular for $x \exp(x) - 1 = 0$ the simplest algorithm described by $x^{(it-1)} + \alpha_{it}$ is the second cheapest one and the algorithm $x^{(it-1)} + \alpha_{it} f\left(x^{(it-1)}\right)^2$ is surprisingly the cheapest algorithm.

In contrast, for two or more dimensions, such spurious behavior is not expected or obtained. A simple optimization of the step size is not enough for an algorithm to converge to a root or a minimum of a given function, since the direction provided by the gradient is crucial. For the two dimensional problems not many new algorithms are found. For instance, for the well-known and quite challenging Rosenbrock function, allowing at most 20 iterations, only Newton's algorithm is found to be feasible. This is not very surprising given the quite restrictive setting considered: single-step methods, modest number of iterations allowed, and restricted choice of step sizes $\alpha_{it} = \pm 2^{-\bar{\alpha}_{it}}$. Note also that the global optimizer did not converge in all cases so that we cannot exclude the existence of feasible algorithms and even thou Newton's algorithm needs less than 10 iterations, the solver did not find the algorithm to be feasible when only 10 iterations are allowed.

4.4 Two-step methods

We first considered the 1-dimensional strongly convex problem $\min_x \exp(x) + x^2$ over $X = [-2, 2]$ with starting point $x^{(0)} = y^{(0)} = 0.1$.

We also considered the minimization (quad2d). The function is convex and one could argue that it is too simple but if we want to find any algorithm that converges within a predefined maximum number of iterations it_{con}, there of course has to exist such an algorithm in first place.

The results can be seen in Fig. 1. We found 13 feasible algorithms for this problem. We allowed at most 10 iterations. The 5 cheapest algorithms did not even require 5 iterations and the 8 other feasible algorithms also converged in under 5 iterations. Nesterov's method is the cheapest one found for this particular problem. In general, one could say that, except for Newton's method (green curve in Fig. 1), all algorithms are alterations of Nesterov's method.

Additionally, we considered the minimization of the convex function (exp2d). The optimimum of f is at $\mathbf{x}^* \approx (-0.257627653, -0.257627653)$. The results can be seen in Fig. 2. Note that this time Newton's algorithm is **not** among the 5 cheapest algorithms for this problem. It is also important to remark that the 5 algorithms all follow the same path, as can be seen in the contour plot shown in Fig. 2. This is caused by the choice of the starting point and by the symmetry of the considered function. The examples show that it is possible for us to discover non-trivial algorithms if the formulation is adjusted appropriately, albeit at high cost of solving the optimization formulations.

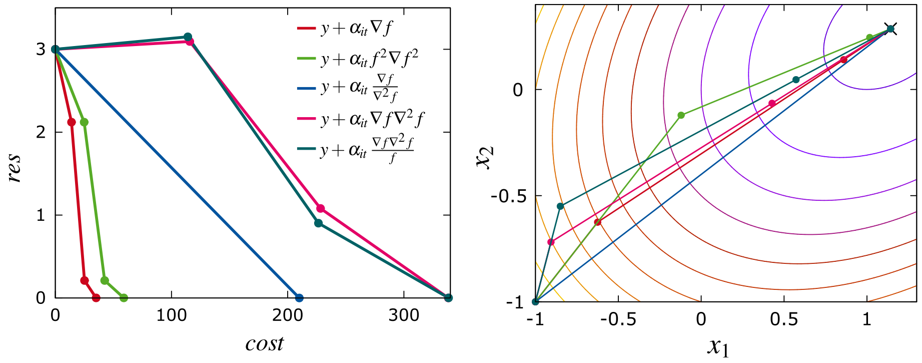

Fig. 1 The five cheapest algorithms for the minimization of $f(x_1, x_2) = (x_1 - 1)^2 + 2x_2^2 - x_1x_2$ over $X = [-1, 2]^2$ with starting point $\mathbf{x}^{(0)} = \mathbf{y}^{(0)} = (-1, -1)$. Nesterov's algorithm (shown in red) is found to be the cheapest. All algorithms seem very similar as they all use gradient information with only small differences in the additional function value multiplicator and the use of Hessian information. Here y stands for the value of the last iteration $x^{(it-1)}$. (Color figure online)

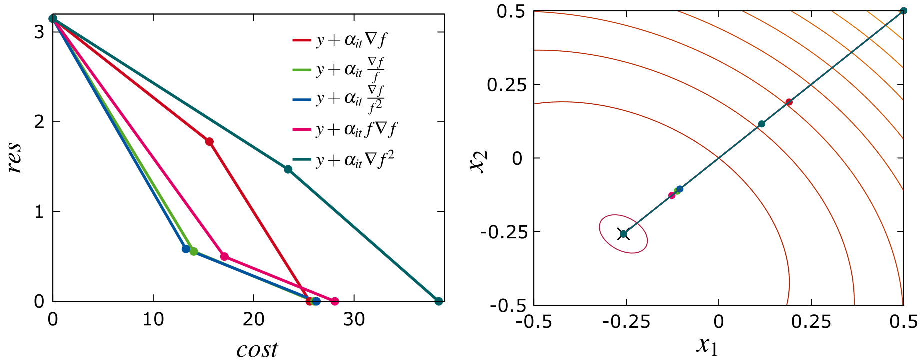

Fig. 2 The five cheapest algorithms for the minimization of $f(x_1, x_2) = \exp(x_1) + x_1^2 + \exp(x_2) + x_1^2 + x_1x_2$ over $[-1, 1]^2$ with starting point $\mathbf{x}^{(0)} = \mathbf{y}^{(0)} = (0.5, 0.5)$. Nesterov's algorithm is the cheapest, shown in red. All algorithms seem again very similar as they all use gradient information with only small differences in the additional function value multiplicator and the use of Hessian information. Note that Newton's algorithm is not among the 5 cheapest. Here y stands for the value of the last iteration $x^{(it-1)}$. (Color figure online)

4.5 Initial conditions

After finding two-step algorithms for the functions mentioned in Sect. 4.4, we investigated the behavior of the 5 cheapest algorithms for several starting points.

Let us first discuss the results for the strongly convex function (convex) and the 5 cheapest algorithms found. We chose 17 different starting points

$$x^{(0)} = y^{(0)} \in \{-2, -1.75, \ldots, 2\}.$$

All five algorithms found for the starting point $x^{(0)} = y^{(0)} = 0.1$ were also feasible for the 17 starting points considered. Nesterov's method is "best" among these in the sense that it was the cheapest in 15 cases and second cheapest in the remaining 2 cases. A similar statement cannot be made on the other 4 algorithms since their ranking w.r.t. cost varied depending on the initial point. Still, none of the 4 other algorithms had significant increases in cost which can probably be explained by the strong convexity of the considered function and the similarity with Nesterov's method manifested in the 4 algorithms.

Next, let us discuss the results for (quad2d) and the 5 cheapest algorithms shown in Fig.1 for starting point $\mathbf{x}^{(0)} = \mathbf{y}^{(0)} = (-1, -1)$. We chose 15 different starting points $\mathbf{x}^{(0)} = \mathbf{y}^{(0)} \in \{(2, 2), (2, 1), (2, 0), (2, -1), (1, 2), (1, 1), (1, 0), (1, -1), (0, 2), (0, 1), (0, 0), (0, -1), (-1, 2), (-1, 1), (-1, 0), (-1, -1)\}$. Nesterov's method and Newton converged for all starting points and Nesterov was the cheapest for every initial condition. The other algorithms did not converge for all starting points or even became infeasible, e.g., the algorithm $y^{(it-1)} + \alpha_{it} f^2 \nabla f^2$ was infeasible for every starting point containing a 0 coordinate. Still, even though the unknown algorithms were infeasible for some starting points, the ranking w.r.t. the cost did not change for the cases where all algorithms converged.

Last, let us discuss the results for (exp2d) and the 5 cheapest algorithms shown in Fig. 2 for starting point $\mathbf{x}^{(0)} = \mathbf{y}^{(0)} = (0.5, 0.5)$. We chose 13 different starting points $\mathbf{x}^{(0)} = \mathbf{y}^{(0)} \in \{(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1), (0.4, 0.3), (0.9, -0.1), (-0.6, -0.8), (-0.3, -0.9)\}$. Here we could observe a behavior that was not quite expected. The algorithms only converged for 7 out of the 13 starting points $\{(-1, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1), (0.4, 0.3)\}$ and not for the rest. Some of these 7 starting points are placed on the diagonal path taken by the algorithms, seen in Fig. 2 and the other starting points seemed to simply provide useful derivative information. The algorithms either were infeasible for the other 6 starting points or the optimizer did not converge in the given time. Likely the infeasibility is due to using the same α and β for both coordinates in the algorithms, which is commonly done. Since the chosen function has symmetrical properties, the algorithms only converge under some specific conditions. In that sense the proposed method gives insight into devising new algorithmic ideas, e.g., different β for different coordinates.

5 Limitations

The proposed formulation is prototypical and a number of challenges arise naturally. As aforementioned, the knowledge of an explicit cost may not be a good approximation for all problems. It is also clear that different objective functions, including, e.g., considerations of memory usage or the inclusion of error correction features, may dramatically effect the results.

We expect the proposed optimization formulation to be very hard to solve and the numerical results confirm this. In particular we expect it to be at least as hard as the original problems. Proving that statement is outside the scope of the manuscript but two arguments are given, namely that the final constraint corresponds to solution of problem and that the subproblems of the optimization problem are the same or at least related to the original problem.

Herein brute force and general-purpose solvers are used for the solution of the optimization problems. It is conceivable to develop specialized numerical algorithms that will perform much better than general-purpose ones due to the specific problem structure. In essence we have an integer problem with linear objective and a single nonlinear constraint. It seems promising to mask the intermediate variables from the optimizer. This would in essence follow the optimization with implicit functions embedded [30] and follow-up work. It is also conceivable to move to parallel computing, but suitable algorithms are not yet parallelized.

In our formulation, f is assumed to be a given function, so that we can apply current numerical methods for the optimization. It is, however, possible to consider multiple instances of the problem simultaneously, i.e., allow f to be among a class of functions. This can be done similar to stochastic optimization [7]. A simple method is to sample the instances of interest (functions f) and optimize for some weighted/average performance of the algorithm. Alternatively, the instances can be parametrized by continuous parameters and the objective in the above formulations replaced with some metric of the parameter distribution, e.g., the expectation of the cost. It is also possible, and likely promising, to consider worst-case performance, e.g., in a min-max formulation [18]. It is also interesting to consider the relation of this work to the theorems developed in [45], in particular that optimality of an algorithm over one class of problems does not guarantee optimality over another class.

6 Future work: possible extensions

In principle, the proposed formulation can be applied to any problem and algorithm. In this section we list illustrative extensions to other problems of interest, starting with relatively straightforward steps and progressing towards exploratory possibilities. In the discussion we try to distinguish between relative extensions that could be easily done and somewhat speculative potential ideas.

6.1 Optimal tuning of algorithms

Many algorithms have tuning parameters. Optimizing these for a given fixed algorithm using similar formulations as presented is straightforward. Of course, alternative proposals exist, e.g., [25].

6.2 Matrices

Regarding the possibility of working with matrices, a formulation for finding algorithms from the family of Quasi–Newton methods could be formulated. The general idea of the Quasi–Newton methods is to update an approximate Hessian matrix with the use of gradient difference $\nabla f(x^{(it)}) - \nabla f(x^{(it-1)})$. Then an approximation of the Hessian matrix is computed by the use of, e.g., Broyden's method or the Davidon-Fletcher-Powell formula, which both can be expressed with one equation.

6.3 Rational polynomial algorithms

A rather obvious generalization is to consider not just a single monomial but rather rational polynomials involving the derivatives. More generally this could be extended to include non-integer and possibly even irrational powers. This would also allow algorithms involving Taylor-series expansion. No conceptual difficulty exists for an optimization formulation similar to the above but the number of variables increases.

6.4 Integral operations

The formulation essentially also allows algorithms that perform integration, if $f^{(j)}$ with $j < 0$ is allowed. Obviously for multivariate programs ($n_x > 1$) the dimensionality needs to be appropriately considered.

6.5 Implicit methods

Implicit methods, e.g., implicit Euler, do not only involve evaluation of derivatives but also the solution of a system of equations at each step. In other words we cannot directly write such methods as monomials in Definition 4. However, it is conceptually relatively easy to adjust the formulation, in particular by changing the update scheme to something along the lines of

$$x^{(it)} = x^{(it-1)} + \alpha \Pi_{j=0}^{j_{max}} \left(f^{(j)}(x^{(it)}) \right)^{\nu_{j,1}}.$$

Computationally, the optimization problems will become substantially harder. The variables $x^{(it)}$ cannot be directly eliminated but rather the equations have to be given to the optimizer or addressed as implicit functions, see [30,38].

6.6 General multistep methods

Explicit multistep methods are those that calculate the current iterate based not only on the immediately previous but rather also on additional preceding steps. Examples include multistep integration methods as well as derivative-free equation solution methods such as secant or bisection. It is straightforward to extend the formulations to allow such methods, e.g., for two-step methods

$$\begin{aligned} x^{(it)} = x^{(it-1)} &+ \alpha_1 \Pi_{j=0}^{j_{max,1}} \left(f^{(j)}(x^{(it-1)}) \right)^{\nu_{j,1}} \\ &+ \alpha_2 \Pi_{j=0}^{j_{max,2}} \left(f^{(j)}(x^{(it-2)}) \right)^{\nu_{j,2}}. \end{aligned}$$

Obviously the number of optimization variables increases accordingly. This can also be used for methods that rely on approximation of derivatives, e.g., Quasi–Newton methods. Similarly, methods such as secant can be encoded by allowing mixed polynomials of $\mathbf{x}$ along with $\mathbf{f}^{(j)}$. Bisection is not captured as this requires if-then-else; however, it should be relatively easy to capture this using integer variables, which fits the proposed MINLP framework. Obviously for *implicit multistep* methods the two preceding formulations can be combined with a formulation along the lines of

$$x^{(it)} = x^{(it-1)} + \sum_{\hat{it}=\hat{it}_{min}}^{\hat{it}_{max}} \alpha_{it,\hat{it}} \Pi_{j=0}^{j_{max,1}} \left(f^{(j)}(x^{(it+\hat{it})}) \right)^{\nu_{j,\hat{it}}}.$$

6.7 Multiple families of algorithms

The article focuses on optimizing with a family of algorithms, described by a parametrized function. A natural question that arises is if multiple families can be considered. One approach is to encode each family in its own optimization problem, solve multiple optimization problems and postprocess the results. Another approach is to devise a function that describes all algorithms, i.e., a more general family of algorithms that encompasses all families. In any case the computational cost may substantially increase.

6.8 Global optimization

A challenge in global optimization is that there are no explicit optimality criteria that can be used in the formulation. The definition of global optimality $f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in X$ can be added to the optimization of algorithm formulation but this results in a (generalized) semi-infinite problem (SIP). There are methods to deal with SIP problems but they are computationally very expensive [29]. Another challenge is that deterministic and stochastic global methods do not follow the simple update using just monomials as in Definition 4. As such, major conceptual difficulties are expected for such algorithms, including an estimation of the cost of calculating the relaxations. Moreover, to estimate the cost of such algorithms the results of ongoing analysis in the spirit of automatic differentiation will be required [10,16] and the cost does not only depend on the values of the derivatives at a point. For instance in αBB [1,27] one needs to estimate the eigenvalues of the Hessian for the domain; in McCormick relaxations [28,42] the cost depends on how many times the composition theorem has to be applied. Recall also the discussion on cost for some local algorithms such as line-search methods.

6.9 Optimal algorithms with optimal steps

Another interesting class are algorithms that include an optimization step within them, such as selection of an optimal step, e.g., [44,46]. We can consider a bilevel formulation of the form

$$\begin{aligned} \min \sum_{it=1}^{it_{con}} & \phi(g(x^{(it)}, z^{(it)})) \\ \text{s.t. } x^{(it)} &= g(x^{(it-1)}, z^{(it-1)}), \quad i = 1, \ldots, it_{con} \\ z^{(it)} &\in \arg\min_{z'} f(x^{(it-1)}, z'), \quad i = 1, \ldots, it_{con}, \end{aligned} \tag{10}$$

describing problems where the next step is given by an optimality condition, e.g., f could describe the least squares formulation for the substep in a generalized Gauss–Newton algorithm in which the outer problem is similar to the formulation above, while the lower-level problem uses a different objective to calculate the optimal step. There exist (quite expensive) approaches to solve bilevel programs with nonconvexities [31]. When the lower level problem is unconstrained regarding f can be avoided, as is the case in generalized Gauss–Newton algorithms, then the problem becomes much simpler. Note that although we have several arg min, they are not correlated, i.e., we do not obtain a n-level optimization problem but rather a bilevel problem with multiple lower level problems. With such a formulation we might be able to prove optimality of one of the three gradient methods described in [44] or discover alternatives. Additionally it would be possible to discover well-known methods such as the nonlinear conjugate gradient method [26], where the line search is described by the arg min operator, or even stochastic methods which often depend on the computation of the maximum expected value of some iterative random variable $\mathcal{X}^{(it)}$. Similarly, it could be possible to find the desired controller algorithm in [17].

6.10 Continuous form

Discrete methods are often reformulated in a continuous form. For instance, Boggs proposed a continuous form of Newton's method [8,9] $\dot{x}(t) = \frac{f(x(t))}{f^{(1)}(x(t))}$. See also the recent embedding of discrete algorithms like Nesterov's scheme in continuous implementations [39,43]. It seems possible to consider the optimization of these continuous variants of the algorithms using similar formulations. Typically discretization methods are used to optimize with such dynamics embedded. Thus, this continuous formulation seems more challenging to solve than the discrete form above. If a particular structure of the problem can be recognized, it could however be interesting, for instance, to apply a relaxation similar to [37].

6.11 Dynamic simulation algorithms

The task here is to simulate a dynamical system, e.g., ordinary differential equations (ODE) $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t))$ along with some initial conditions $\mathbf{x}(t = 0) = \mathbf{x}^{init}$ for a given time interval $[0, t_f]$. We need to define what is meant by a feasible algorithm. A natural definition involves the difference of the computed time trajectory from the exact solution, which is not known, and thus does not yield an explicit condition. One should therefore check the degree to which the resulting approximate curve, possibly after interpolation, satisfies the differential equation over this time interval.

6.12 Dynamic optimization algorithms

Dynamic optimization combines the difficulties of optimization and dynamic simulation. Moreover, it results in a somewhat amusing cyclic problem: we require an algorithm for the solution of dynamic optimization problems to select a dynamic optimization algorithm. As aforementioned, this is not prohibitive, e.g., in the offline design of an algorithm to be used online.

6.13 Algorithms in the limit

Considering algorithms that provably converge to the correct solution in the limit makes the optimization problem more challenging. The infinite-dimension formulation (1) is in principle applicable with the aforementioned challenges. If the finite (parametrized) formulation (5), was directly applied, an infinite number of variables would have to be solved for. In such cases one could test for asymptotic self-similarity of the algorithm behavior as a way of assessing its asymptotic result.

6.14 Quantum Algorithms

A potential breakthrough in computational engineering would be realized by quantum computers. These will require new algorithms, and there are several scientists that are developing such algorithms. It would thus be of extreme interest to consider the extension of the proposed formulation to quantum algorithms and/or their real-world realizations. This may result in "regular" optimization problems or problems that need to be solved with quantum algorithms themselves.

7 Conclusions

An MINLP formulation is proposed, that can devise optimal algorithms (among a relatively large family of algorithms) for several prototypical problems, including solution of nonlinear equations and nonlinear optimization. Simple numerical case studies demonstrate that well-known algorithms can be identified, and so can new ones. We argue that the formulation is conceptually extensible to many interesting classes of problems, including quantum algorithms. Substantial work is now needed to develop and implement these extensions so as to numerically devise optimal algorithms for interesting classes of challenging problems where such algorithms are simply not known. Also, the similarity to model-reference adaptive control (MRAC) and internal model control (IMC) can be further explored in the future. The optimization problems obtained can, however, be very challenging and no claim is made that optimal algorithm discovery will be computationally cheap. However, in addition to the theoretical interest, there are certainly applications. Finding *guaranteed* optimal algorithms for a given problem implies understanding/classifying the difficulty of this problem. And it will certainly be worthwhile to automatically devise algorithms offline, that will be used to solved problems online.

Acknowledgements IGK and AM are indebted to the late C.A. Floudas for bringing them together. Fruitful discussions with G.A. Kevrekidis and the help of C.W. Gear with the continuous formulation are greatly appreciated. The anonymous reviewers provided helpful feedback that resulted in an improved manuscript.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

Appendix

A Solution of system of equations

For illustration purposes we will discuss the solution of system of equations using only zero and first derivatives. The zero derivative $f^{(0)}\left(\mathbf{x}^{(it)}\right)$ is a vector of the same size as the point $\mathbf{x}$. By definition, the required polynomial expression of the derivative $\left(f^{(0)}\left(\mathbf{x}^{(it)}\right)\right)^{\nu_k}$ is calculated element by element. Thus, we can equivalently rewrite each element if of the vector $\left(\mathbf{f}^{(0)}\left(\mathbf{x}^{(it-1)}\right)\right)^{\nu_k}$ as $\sum_{k=k_{min}}^{k_{max}} y_0^k \left(f_{if}^{(0)}\left(\mathbf{x}^{(it-1)}\right)\right)^k$ along with a constraint $\sum_{k=k_{min}}^{k_{max}} y_0^k = 1$. In contrast, the first derivative is the Jacobian matrix $\mathbf{J}$ with elements $\left.\frac{\partial f_{if}}{\partial x_{ix}}\right|_{\mathbf{x}^{(it)}}$. In principle the inverse can be written analytically but this is only realistic for small-scale systems. For systems of substantial sizes, it is better to calculate the direction via the solution of a system of equations.

$$\mathbf{J}\left(\mathbf{x}^{(it-1)}\right)\Delta\mathbf{x} = \left(\mathbf{f}^{(0)}\left(\mathbf{x}^{(it-1)}\right)\right)^{\nu_k},$$

the solution of which gives us the direction $\Delta\mathbf{x}$ to be taken if the optimizer selects the inverse of the Jacobian, i.e., sets $\nu_1 = -1$. Similarly for other values of ν_1 we have to calculate other steps via explicit matrix multiplication or via similar systems of equations. As aforementioned, the equations can be explicitly or implicitly written, see [30] and its extensions [38].

B Alternative to obtain a regular optimization problem

For the sake of simplicity, an alternative is presented to obtaining a finite number of variables in the optimal control problem. Again a maximal number of iterations it_{max} is considered and the cost is penalized by the residual, e.g.,

$$\begin{aligned} &\min_{g,x^{(it)}} \sum_{it=1}^{it_{max}} \left|f\left(x^{(it)}\right)\right| \phi\left(g\left(x^{(it)}\right)\right) \\ &\text{s.t. } x^{(it)} = g\left(x^{(it-1)}\right), \quad it = 1, 2, \ldots, it_{max} \end{aligned} \tag{11}$$

$$\left|f\left(x^{(it_{max})}\right)\right| < \varepsilon, \tag{12}$$

where again the last constraint ensures convergence. Without it, if algorithms with zero cost exist, then these will be chosen even if they are infeasible; they will give a zero objective value without meeting convergence.

C Obtaining a regular MINLP

The finite problem proposed contains terms $\left(f^{(j)}(x^{(it-1)})\right)^{\nu_j}$. We thus exponentiate a potentially negative real-valued $(f^{(j)}(x^{(it-1)}))$ to the power ν_j. At feasible points ν_j is integer and thus the operation well-defined. However, in intermediate iterations of the optimization, e.g., in branch-and-bound, ν_j may be real-valued and thus the operation is not defined. Consequently, the optimization formulation cannot be directly solved using standard MINLP solvers. It is however, relatively easy to reformulate as an MINLP with linear dependence on the integer variables. One way is to introduce for each j as many binary variables y_j^k as we have potential values for ν_j and enforce that exactly one of the binaries is one (special ordered set of type 1). Then the term $f^{(j)}(x^{(it-1)})$ is equivalently rewritten as $\sum_{k=k_{min}}^{k_{max}} y_j^k \left(f^{(j)}(x^{(it-1)})\right)^k$ along with a constraint $\sum_{k=k_{min}}^{k_{max}} y_j^k = 1$.

References

1. Adjiman, C.S., Floudas, C.A.: Rigorous convex underestimators for general twice-differentiable problems. J. Glob. Optim. **9**(1), 23–40 (1996)
2. Arya, S., Mount, D.M., Netanyahu, N., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. In: Proceedings of the 5th ACM-SIAM Symposium Discrete Algorithms, pp. 573–582 (1994)
3. Bacher, R.: Automatic generation of optimization code based on symbolic non-linear domain formulation. In: Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation, pp. 283–291. ACM (1996)
4. Bain, S., Thornton, J., Sattar, A.: Methods of automatic algorithm generation. In: PRICAI 2004: Trends in Artificial Intelligence, pp. 144–153. Springer (2004)
5. Biegler, L.T.: Nonlinear programming: concepts, algorithms, and applications to chemical processes. MPS-SIAM Series on Optimization. SIAM-Society for Industrial and Applied Mathematics (2010)
6. Bientinesi, P.: Mechanical derivation and systematic analysis of correct linear algebra algorithms. Ph.D. Thesis, Graduate School of The University of Texas at Austin (2006)
7. Birge, J.R., Louveaux, F.: Introduction to Stochastic Programming. Springer, Berlin (1997)
8. Boggs, P.T.: The solution of nonlinear systems of equations by a-stable integration techniques. SIAM J. Numer. Anal. **8**, 767–785 (1971)
9. Boggs, P.T.: The convergence of the Ben-Israel iteration for nonlinear least squares problems. Math. Comput. **30**, 512–522 (1976)
10. Bompadre, A., Mitsos, A.: Convergence rate of McCormick relaxations. J. Glob. Optim. **52**(1), 1–28 (2012)
11. Brooke, A., Kendrick, D., Meeraus, A.: GAMS: A User's Guide. The Scientific Press, Redwood City (1988)
12. Byrd, R.H., Nocedal, J., Waltz, R.A.: KNITRO: An Integrated Package for Nonlinear Optimization, vol. 83, pp. 35–59. Springer, Berlin (2006)
13. Coelho, C.P., Phillips, J.R., Silveira, L.M.: Robust rational function approximation algorithm for model generation. In: Proceedings of the 36th Annual ACM/IEEE Design Automation Conference, pp. 207–212. ACM (1999)
14. Deuflhard, P.: Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms, vol. 35. Springer, Berlin Heidelberg (2004)
15. Drori, Y., Teboulle, M.: Performance of first-order methods for smooth convex minimization: a novel approach. Math. Program. **145**(1), 451–482 (2014)

16. Du, K.S., Kearfott, R.B.: The cluster problem in multivariate global optimization. J. Glob. Optim. **5**(3), 253–265 (1994)
17. Economou, C.G.: An operator theory approach to nonlinear controller design. Ph.D. Thesis, California Institute of Technology Pasadena, California (1985)
18. Falk, J.E., Hoffman, K.: A nonconvex max-min problem. Naval Res. Logist. **24**(3), 441–450 (1977)
19. Garber, B.A., Hoeflinger, D., Li, X., Garzaran, M.J., Padua, D.: Automatic generation of a parallel sorting algorithm. In: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, pp. 1–5. IEEE (2008)
20. Kim, D., Fessler, J.A.: Optimized first-order methods for smooth convex minimization. Math. Program. **159**(1), 81–107 (2016)
21. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection, vol. 1. MIT press, Cambridge (1992)
22. Koza, J.R.: Genetic programming ii: Automatic discovery of reusable subprograms. Cambridge, MA, USA (1994)
23. Kuhner, M., Burgoon, D., Keller, P., Rust, S., Schelhorn, J., Sinnott, L., Stark, G., Taylor, K., Whitney, P.: Automatic algorithm generation (2002). US Patent App. 10/097,198
24. Lessard, L., Recht, B., Packard, A.: Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints. ArXiv e-prints (2014)
25. Li, Q., Tai, C., E, W.: Dynamics of stochastic gradient algorithms. arXiv:1511.06251 (2015)
26. Luenberger, D.G.: Introduction to Linear and Nonlinear Programming, vol. 28. Addison-Wesley, Reading (1973)
27. Maranas, C.D., Floudas, C.A.: A global optimization approach for Lennard-Jones microclusters. J. Chem. Phys. **97**(10), 7667–7678 (1992)
28. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: part I. Convex underestimating problems. Math. Program. **10**(1), 147–175 (1976)
29. Mitsos, A.: Global optimization of semi-infinite programs via restriction of the right hand side. Optimization **60**(10–11), 1291–1308 (2011)
30. Mitsos, A., Chachuat, B., Barton, P.I.: McCormick-based relaxations of algorithms. SIAM J. Optim. **20**(2), 573–601 (2009)
31. Mitsos, A., Lemonidis, P., Barton, P.I.: Global solution of bilevel programs with a nonconvex inner program. J. Glob. Optim. **42**(4), 475–513 (2008)
32. Nemirovsky, A., Yudin, D.: Problem Complexity and Method Efficiency in Optimization. J. Wiley, New York (1983)
33. Parada, L., Sepulveda, M., Herrera, C., Parada, V.: Automatic generation of algorithms for the binary knapsack problem. In: Evolutionary Computation (CEC), 2013 IEEE Congress on, pp. 3148–3152. IEEE (2013)
34. Ricart, G., Agrawala, A.K.: An optimal algorithm for mutual exclusion in computer networks. Commun. ACM **24**(1), 9–17 (1981)
35. Ruuth, S.: Global optimization of explicit strong-stability-preserving Runge–Kutta methods. Math. Comput. **75**, 183–207 (2006)
36. Ruuth, S., Spiteri, R.: High-order strong-stability-preserving Runge–Kutta methods with downwind-biased spatial discretizations. SIAM J. Numer. Anal. **42**(3), 974–996 (2004)
37. Sager, S., Bock, H.G., Reinelt, G.: Direct methods with maximal lower bound for mixed-integer optimal control problems. Math. Program. **118**(1), 109–149 (2009)
38. Stuber, M.D., Scott, J.K., Barton, P.I.: Convex and concave relaxations of implicit functions. Optim. Methods Softw. **30**(3), 424–460 (2015)
39. Su, W., Boyd, S., Candes, E.J.: A Differential Equation for Modeling Nesterov's Accelerated Gradient Method: Theory and Insights. ArXiv e-prints (2015)
40. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Math. Program. **103**(2), 225–249 (2005)
41. Tozoni, D.C., Rezende, P.J.D., Souza, C.C.D.: Algorithm 966: a practical iterative algorithm for the art gallery problem using integer linear programming (2016)
42. Tsoukalas, A., Mitsos, A.: Multivariate McCormick relaxations. J. Glob, Optim. **59**(2–3), 633–662 (2014)
43. Wibisono, A., Wilson, A.C., Jordan, M.I.: A Variational Perspective on Accelerated Methods in Optimization. ArXiv e-prints (2016)
44. Wibisono, A., Wilson, A.C., Jordan, M.I.: A variational perspective on accelerated methods in optimization. arXiv preprint arXiv:1603.04245 (2016)
45. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evolut. Comput. **1**(1), 67–82 (1997)

46. Zhang, Y., Chen, X., Zhou, D., Jordan, M.I.: Spectral methods meet em: a provably optimal algorithm for crowdsourcing. In: Advances in neural information processing systems, pp. 1260–1268 (2014)