On the approximation of separable non-convex optimization programs to an arbitrary numerical precision

Claudio Contardo, Sandra Ulrich Ngueveu

▶ **To cite this version:**

Claudio Contardo, Sandra Ulrich Ngueveu. On the approximation of separable non-convex optimization programs to an arbitrary numerical precision. 2025. hal-03336022v3

HAL Id: hal-03336022
https://hal.science/hal-03336022v3

Preprint submitted on 21 Jan 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On the approximation of separable non-convex optimization programs to an arbitrary numerical tolerance

Claudio Contardo[1] Sandra Ulrich Ngueveu[2]

[1]Concordia University, Montreal, Canada
[2]Université de Toulouse, CNRS, INP, LAAS, Toulouse, France
January 21, 2025

Abstract

We consider the problem of minimizing the sum of a series of univariate (possibly non-convex) functions on a polyhedral domain. We introduce an iterative method with optimality guarantees to approximate this problem to an arbitrary numerical tolerance. At every iteration, our method replaces the objective by a piecewise linear relaxation to compute a dual bound. Since the polyhedral domain in our method remains unchanged, a primal bound is computed by evaluating the cost function on the solution provided by the relaxation. If the difference between these two values is deemed as not satisfactory, the relaxation is locally tightened with an objective-driven refinement procedure, that computes an optimal domain partitioning and the process repeated. By keeping the scope of the update local, the computational burden is only slightly increased from iteration to iteration. The convergence of the method is assured under very mild assumptions, and no NLP nor MINLP solver/oracle is required to ever be invoked to do so. As a consequence, our method presents very nice scalability properties and is little sensitive to the desired tolerance. We provide a formal proof of the convergence of our method, and assess its efficiency in approximating the non-linear variants of five problems: the transportation problem, the uncapacitated facility location problem, the multicommodity flow problem, the multi-commodity network design problem, and the continuous knapsack problem. Our results indicate that the overall performance of our method is competitive to three state-of-the-art mixed-integer nonlinear solvers, often performing better. It also scales better than a naive variant of the method that avoids performing successive iterations in exchange of solving a much larger mixed-integer linear program.

1 Introduction

We consider the problem of solving the following mixed-integer non-linear program (MINLP) with a separable objective:

$$x \in \arg\min \left\{ \sum_{i=1}^{n} f_i(x_i) : Ax = b, x \in \mathbb{R}_+^{n-p} \times \mathbb{Z}_+^p \right\}, \tag{1}$$

where the functions $f_i : \mathbb{R} \longrightarrow \mathbb{R}, i = 1 \dots n$ are piecewise differentiable. For the sake of simplicity, we also assume that the problem is well defined and that it admits an optimal solution (although perhaps not an unique one). If $(f_i)_{i=1\dots n}$ are all affine linear functions, the problem is a conventional mixed-integer linear problem (MILP) for which state-of-the-art algorithms and commercial software can scale and solve problems with millions of variables and constraints (IBM, 2020). This is no longer true when some of the functions f_i are not linear. While it is possible to efficiently handle problems for some specific forms of f_i —namely when they are quadratic and/or convex (IBM, 2020; Stellato et al., 2020)—, general forms of the functions f_i make the optimization problems much less tractable.

One possible way to approximate problem (1) is by replacing functions f_i by piecewise linear functions. This procedure results in a MILP with additional binary variables, and can be tackled using state-of-the-art machinery from the integer programming literature. This is perhaps the most efficient way known to handle problems with this structure. By properly choosing the piecewise linear approximations, this approach can provide guarantees on the quality of the solutions achieved. These guarantees, however, come at the extent of potentially very large piecewise linear approximations, and remain only practical for very rough numerical tolerances.

This article addresses the issue of solving problem (1) to an arbitrary numerical tolerance by solving a series of piecewise linear relaxations of the problem in a way such that the tractability of the resulting MILP is not compromised along the process. Our method relies on the existence of a tractable piecewise linear relaxation for a reasonably good (but probably not good enough) tolerance to derive primal and dual bounds, and on an objective-driven refinement method used to achieve a better numerical precision by tightening the relaxation. By objective-driven we mean that our method maintains primal feasibility at all times, but tightens the approximations from one iteration to the next by reducing the maximum error at the objective level. The key in the success of our method lies in the fact that the refinement procedure focuses on the improvement of the approximation precision by design, has very local scope, and the successive relaxations, while becoming tighter, do not lose their tractability, allowing for the primal and dual bounds to converge quickly.

The remainder of this manuscript is organized as follows. In Section 2 we present a literature review that focuses on the numerical approximation of mixed integer non-linear programs using piecewise linear approximations. In Section 3 we make a brief description of `LinA.jl`, a Julia package that finds minimal piecewise linear approximations to functions with error guarantees. In Section 4 we present our method and provide the formal background to justify its convergence. In Section 5 we describe the application of our method to approximating the non-linear variants of five optimization problems relevant in practice: the transportation problem (TP), the uncapacitated facility location problem (UFLP), the multi-commodity flow problem (MCFP), the multicommodity network design problem (MCNDP), and the continuous knapsack problem (CKP). We also provide computational evidence of the efficiency of our method. In Section 6 we provide a brief discussion about additional computational experiments. In Section 7 we discuss some limitations of our method as they become apparent in our computational campaign. Section 8 concludes this manuscript.

2 Related works

Two problems need to be addressed when building MILP-based approximations or relaxations of nonconvex MINLPs with a predefined accuracy : 1) that of obtaining good piecewise linear approximations of the nonlinear functions; and 2) that of efficiently constructing and solving the resulting MILP.

The quality of piecewise linear approximations is typically evaluated using two conflicting criteria: the approximation error evaluated with a relevant metric, and the size of the approximation measured as the number of linear pieces (Ertel and Fowlkes, 1976). To ensure that the desired MINLP accuracy will be achieved, a bound is set on pointwise approximation errors, i.e., that can be expressed in function of the maximal difference between each nonlinear function and its approximation (Geißler et al., 2012). In many cases, compact MILP models with a low number of binary/integer variables lead to faster computing times thanks to the reduced combinatorial explosion of an enumeration tree. Therefore, it is of interest to obtain (near-)optimal piecewise linearizations with respect to the objective of minimizing the number of linear pieces given a predefined pointwise error bound. Among the few publications that tackle this version of the piecewise linearization problem with formal models and exact algorithms to ensure optimality of its solutions, Rebennack and Kallrath (2015) and Rebennack and Krasko (2019) showed that distributing breakpoints freely and allowing shifts from the nonlinear function at breakpoints leads to an order of magnitude less linear pieces compared to

equidistant breakpoints that interpolate the nonlinear univariate function. Ngueveu (2019) authorizes discontinuity in the piecewise linear function even if the original nonlinear function is continuous, yielding an additional degree of freedom to obtain a breakpoint system of equal or less linear segments. Codsi et al. (2021) propose a geometric approach that can solve the problem in quasi-logarithmic time on a very broad class of pointwise error metrics.

Modeling piecewise linear functions in MILP requires additional binary variables and constraints. Various representations have been studied for example by Vielma and Nemhauser (2011), Huchette and Vielma (2019), Hwang and Huang (2021) including ones where the number of binary variables increases only logarithmically with the number of linear pieces. In Vielma et al. (2010) the authors review the literature on modeling piecewise linear functions in MILPs, perform a computational analysis between the different techniques and provide an unifying view of them, including the use of lower semi-continuous piecewise functions.

For general nonconvex MINLP with a linear objective-function and nonlinear constraints —which can be obtained after reformulation of any nonconvex MINLP with a nonlinear objective-function— Geißler et al. (2012) present a general methodology to construct a mixed integer piecewise polyhedral (MIP) relaxation of a MINLP, instead of a mixed integer piecewise linear approximation. It requires piecewise linear segments that interpolate the nonlinear functions at the breakpoints and the calculation of the maximum linearization error for each linear piece. The MIP relaxation produces lower bounds for the MINLP. Then, the authors use an NLP solver to produce feasible solutions for the MINLP once its integer variables have been fixed to their values in the MIP relaxation solution. As a consequence, it is straightforward to implement a branch-and-bound algorithm to solve the MINLP, which can prove optimality and prune infeasible or suboptimal parts of the search tree by using MILP techniques.

The drawback of any MINLP solution method based on piecewise linear approximations is that small approximation errors lead to large MILPs, which become difficult to solve. Burlacu et al. (2020) build on the work of Geißler et al. (2012) and develop an iterative algorithm to find a global optimal solution of the MINLP by solving a series of MILP relaxations with gradually increasing accuracy, based on piecewise linear functions that are adaptively refined from one iteration to another. A critical component concerns the way the piecewise linear functions are defined and their refinement procedure. The authors need piecewise linear functions that interpolate the nonlinear function at the breakpoints and that completely contain the graph of the function. They provide rather general convergence conditions for MINLP solution algorithms that rely on the adaptive refinement of their piecewise linear relaxations. They show that the refinement strategy adding solely points with maximal approximation error on a simplex does not fulfill these conditions and may not converge in certain cases. In contrast, the refinement strategy adding linearization breakpoints on the longest edge of a simplex, such as the classical longest-edge bisection, fulfills these convergence conditions and therefore is suitable for the solution framework proposed which then converges in a finite number of iterations.

Burlacu (2021) extends the iterative algorithm of Burlacu et al. (2020) with another refinement strategy for n-dimensional simplices: the generalized red refinement introduced by Freudenthal (1942). Their procedure is to some extent an n-dimensional generalization of the well-known red-green refinement (Bank et al., 1983), which is used for two-dimensional simplices. However, for the one-dimensional domains we focus on in this paper, i.e., univariate functions, the red refinement and the longest edge refinement are identical and simply split the domain of the active linear pieces in two equal halves by adding, midway through the domain, a breakpoint that interpolates the function. For a more comprehensive survey of refinements of simplicial meshes the reader is referred to Bey (2000). Burlacu (2021) and Burlacu et al. (2020) do not compute any upper bounds, and do not require NLP solvers for such task. However, the authors assume that there is an oracle that optimizes the difference between a nonlinear and linear function over a simplex, in order to compute the linearization errors resulting from the piecewise linear function refinements. Such an oracle may be an NLP solver if the solution of an analytical formula is not available.

The sequential convex MINLP (SC-MINLP, D'Ambrosio et al., 2009, 2012) method approximates

nonlinear problems with separable non-convexities by relaxing the concave parts of the objective by piecewise linear relaxations. The convex parts of the objective are left as is, and dealt with convex optimization solvers. Then, in D'Ambrosio et al. (2019) the authors consider a perspective reformulation of the MINLPs that the authors then strengthen via perspective cuts. Finally, in Trindade et al. (2023) the authors perform a theoretical and computational study of several perspective reformulations for MINLPs with separable nonconvexities. Their results show that the so-called multiple-choice model proves stronger than the original incremental model when strengthened by perspective cuts on problems with nonlinear convexities.

For MINLP with polynomial functions, Nagarajan et al. (2019) propose an adaptive non-uniform partitioning of the domain of variables instead of uniform refinements such as Burlacu (2021)'s. To that end, instead of simply splitting each active partition of the domain of a variable into two equal halves, Nagarajan et al. (2019) split it into three, the size of the middle partition being defined by a user parameter denoted Δ. A Δ value of 8 for instance (default value used by the authors), imposes the size of the new middle partition to be one quarter of the size of the active partition. The authors provide a lemma stating that the value of the lower bound monotonically increases with successive partition refinements. However, there is no proof that the algorithm converges in a finite number of iterations. For solving MINLPs to global optimality the complete MILP-based method proposed by Nagarajan et al. (2019) also integrates domain reduction techniques based on sequential optimization-based bound-tightening (OBBT), construction of piecewise convex relaxations using the partitioned domain, their solution via outer-approximation to obtain updated lower bounds, and the solution of NLPs to obtain new local optimal solutions. The computational evaluation showed that the approach could help solving instances not solvable by previous state-of-the-art solvers. Best known optimality gaps could also be reduced on unsolved instances. The resulting algorithm is available as a global solver for nonconvex MINLPs in a Julia package named `Alpine.jl` (Nagarajan et al., 2016, 2019).

Exact solution methods that solve an instance of an NP-hard problem as a series of smaller instances of the same NP-hard problem have been investigated recently in relation to decremental relaxations and sampling mechanisms. Remarkable examples include the family of methods referred to as dynamic discretization discovery (DDD) for service network design problems. DDD methods are applied on time-indexed models for routing and scheduling problems (Boland et al., 2017, 2019; Marshall et al., 2021), and rely on reducing the sizes of the time-expanded networks by aggregating nodes and arcs. Column generation methods for vehicle routing have also leveraged the use of similar approaches to alleviate the computational burden of solving elementary shortest path problems for pricing new columns. The works of Righini and Salani (2008); Martinelli et al. (2014) show that it is possible to achieve elementary paths by considering partial elementarity in an escalated way. All these works share similarities with our method and with those described in the previous paragraphs: the reformulation of a problem onto an easier/smaller problem capable of providing dual bounds but potentially infeasible solutions, and a refinement mechanism that will find and repair these infeasibilities.

3 `LinA.jl` in a nutshell

In this section we provide a brief description of `LinA.jl`, a Julia implementation of the greedy algorithm of Codsi et al. (2021) for building minimal piecewise linear approximations of nonlinear functions with approximation guarantees. Although `LinA.jl` can handle piecewise linear approximations that are upper bounding, lower bounding, or successive combinations of both, we will describe its application to the case of lower bounding approximations, which is what our algorithm needs.

Let us consider a nonlinear function $f : [x_0, x_f] \longrightarrow \mathbb{R}$. Let us consider another function $g : [x_0, x_f] \longrightarrow \mathbb{R}$ such that $g(x) \leq f(x)$ for every $x \in [x_0, x_f]$. Function g is said to be a *lower bounding approximation* of f in the interval $[x_0, x_f]$. The area between the two functions in the said interval is said to be a *corridor*. Take for instance the example illustrated in Figure 1, with

$f(x) = \sin(\pi x/2) - 0.05x + (0.3x)^2 + 5$ in the interval $[0, 8]$ depicted in a solid red line, the lower bounding approximation $g(x) = 0.8f(x)$ (meaning using a relative tolerance of 20%) depicted in a dashed red line, and the corridor between the two depicted in light blue.

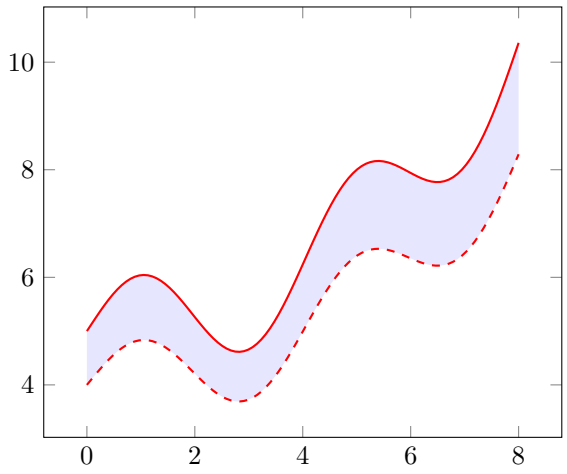


Figure 1: Example of function f, lower bounding approximation g and corridor

The method proposed by Codsi et al. (2021) will proceed as follows. The authors design a method $L(f, g, x_0, x_f)$ that constructs the longest line segment starting at $x = x_0$ that fits into the corridor between f and g. Suppose that the resulting line segment ends at $x_1 \in]x_0, x_f[$. Next, the method proceeds to execute $L(f, g, x_1, x_f)$ to construct the longest line segment starting now from x_1. This process is applied iteratively until reaching the point where $x_k = x_f$ for a certain iteration k. The authors prove (see Codsi et al., 2021, Section 3.2) that this greedy procedure constructs —for a given corridor— the minimal piecewise approximation fitting the corridor, where minimality is defined with respect to the number of line segments.

To further illustrate the method, let us consider the same example using $f(x) = \sin(\pi x/2) - 0.05x + (0.3x)^2 + 5$, $g(x) = 0.8f(x)$ in the interval $[0, 8]$. The solution to $L(f, g, 0, 8)$ will return the longest line segment starting at $x = 0$, namely the line line $l_1(x) = -0.13x + 4.977$ extending until $x_1 = 3.71$. Then, the solution to $L(f, g, 3.71, 8)$ will return the longest line segment starting at $x_1 = 3.71$ and will be given by the equation $l_2(x) = 0.708x + 2.917$ that extends until $x_2 = x_f = 8$. The resulting piecewise linear approximation is given in Figure 2. The two solid blue line segments correspond to the minimal piecewise linear approximation fitting the corridor. Sometimes we will refer to this approximation as a *piecewise linear relaxation* to highlight the fact that it approximates function f by below (in a minimization setting). It is worth mentioning that, since $g = 0.8f$, `LinA.jl` provides in this case a piecewise linear relaxation that is at most 20% lower than f.

One can think of function g as the allowed deviation of the piecewise linear approximation from function f. Assuming that $f(x) \geq 0$ for every $x \in [x_0, x_f]$, `LinA.jl` can be seen as taking $\epsilon \in]0, 100[$ as an input parameter and return a minimal piecewise linear approximation that is at most ϵ% lower than f. Unfortunately, the number of linear pieces, while minimum, can grow quite quickly. For instance, using the same example as before, if one uses $\epsilon = 10\%, 1.0\%, 0.1\%, 0.01\%, 0.001, 0.0001\%$ the number of linear pieces constructed by `LinA.jl` are of 5, 14, 43, 133, 418, and 1303, respectively.

The method is not only shown by the authors to be efficient in terms of the number of line segments, but also in terms of computational efficiency: it performs a dichotomy search on the interval for functions f, g that do not present changes in concavity (i.e. that are either convex or concave). For general functions with changes in concavity, the authors show that the problem can be posed as a data fitting problem (Tomek, 1974a,b) and solved using an ad-hoc method like the one of O'Rourke

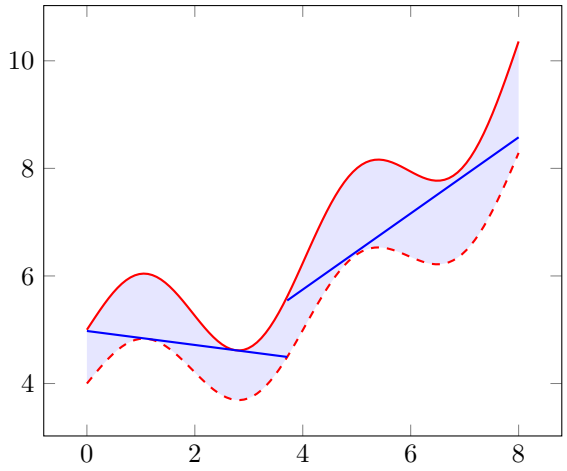


Figure 2: Example of minimal piecewise linear approximation obtained using `LinA.jl`

(1981). In practice, the authors show that one can benefit from the dichotomy search for most of the procedure, and execute the more expensive data fitting method only around the inflection points.

4 The proposed method

In this section we present our iterative method to solve problem (1) to a given relative tolerance $\epsilon > 0$, this is we aim at obtaining a feasible solution $\mathtt{x}$ of (1) whose objective value $z(\mathtt{x}) = \sum_{i=1}^{n} f_i(\mathtt{x}_i)$ is such that $z(\mathtt{x}) - z^* \leq \epsilon |z(\mathtt{x})|$, where z^* is the optimal value of problem (1).

To fix ideas, let us assume that we are given, in addition to problem (1) and ϵ, an initial tolerance $\epsilon_0 \geq \epsilon$, and a minimum interval size $\delta > 0$. The exact meaning of these two additional parameters will become apparent in the description of our method.

4.1 High-level description of the algorithm

At every iteration, our method replaces the objective by a piecewise linear relaxation of the functions f_i to compute a dual bound. Given that the constraint set remains the same for the original problem and the relaxation, a primal bound is readily available by simply evaluating the nonlinear cost function on the solution provided by the relaxation. If the difference between these two values is deemed as not satisfactory, the relaxation is locally tightened with an *objective-driven refinement procedure*, that computes an *optimal domain partitioning* and the process repeated. By keeping the scope of the update local, the computational burden is only slightly increased from iteration to iteration. The convergence of the method is assured under very mild assumptions, and no NLP nor MINLP solver/oracle is required to ever be invoked to do so. As a consequence, our method presents very nice scalability properties and is little sensitive to the desired tolerance.

We provide a high-level description of our algorithm in the pseudo-code Algorithm 1. In the remainder of this section we describe the distinctive components that differentiate it from the state-of-the-art, in particular the objective-driven tightening mechanism key in the success of our method.

4.2 Objective-driven initialization

We define a linear piece as a tuple $l = (g, \alpha, \beta, x_0, x_f, r)$ with $x_0 \leq x_f$, representing the linear function $\alpha x + \beta$ that approximates function g within a tolerance of r in the interval $[x_0, x_f]$. The linear piece

Algorithm 1 Iterative piecewise linear bounding

Input: Problem (1), tolerances $\epsilon, \epsilon_0, \delta$
Output: Feasible solution x^*, dual bound $z^l(x^*)$, primal bound $z(x^*)$ s.t. $z(x^*) - z^l(x^*) \leq \epsilon|z(x^*)|$

```
{f_i^l : i = 1...n} ← initial piecewise linear relaxation with tolerance ε_0        ▷ Section 4.2
while true do
    Solve min_x{Σ_i f_i^l(x_i) : Ax = b, x ∈ R_+^{n-p} × Z_+^p}, let x* be an optimal solution   ▷ Section 4.3
    Let z^l(x*) ← Σ_i f_i^l(x_i*), and z(x*) ← Σ_i f_i(x_i*)
    if z(x*) − z^l(x*) ≤ ε|z(x*)| then        ▷ The desired convergence has been reached
        return (x*, z^l(x*), z(x*))
    else
        Refine {f_i^l : i = 1...n}                                                  ▷ Section 4.4
    end if
end while
```

is said to be a *linear relaxation of g in the interval* $[x_0, x_f]$ if $\alpha x + \beta \leq g(x)$ for every $x \in [x_0, x_f]$. A piecewise linear relaxation of g in an interval $[x_0, x_f]$ is a set $\{(g, \alpha^k, \beta^k, x_0^k, x_f^k, r^k) : k = 1 \dots \kappa\}$ of linear relaxation pieces such that $x_0^1 = x_0, x_f^\kappa = x_f, x_0^i = x_f^{i-1}$ for every $i = 2 \dots \kappa$.

Our method starts by finding an initial piecewise linear relaxation for each function f_i covering the domain of variable x_i (that we assume to be bounded) for the initial tolerance ϵ_0. To that end, we make use of the greedy algorithm described in Section 3, introduced in Codsi et al. (2021), to compute in a negligible computing time (mere milliseconds), optimal (in terms of size) piecewise linear relaxations for continuous univariate nonlinear functions given a bound on the pointwise approximation error allowed. A Julia implementation of the algorithm is available via a package named `LinA.jl`, publicly available at `https://github.com/LICO-labs/LinA.jl`. Note that `LinA.jl` constructs piecewise linear relaxations that are not necessarily continuous and that do not necessarily interpolate the nonlinear functions (indeed, they almost never do). For a given linear piece l, we denote $\epsilon(l)$ the relative tolerance associated with that piece, that initially takes the value ϵ_0 uniformly.

4.3 MILP modeling and solution

We modify problem (1) by replacing the functions $f_i, i = 1 \dots n$ in the objective by their respective piecewise linear relaxations. This results in a modified MILP that approximates problem (1) by providing a solution that is feasible w.r.t. the set of constraints $\{Ax = b, x \in \mathbb{R}_+^{n-p} \times \mathbb{Z}_+^p\}$ and that provides a dual bound for (1). Note that there is not an unique way of writing the modified MILP. In Vielma et al. (2010) the authors review the literature in piecewise linear approximations and propose novel small-sized (in terms of the number of linear pieces) models. In Huchette and Vielma (2019), the authors introduce a refined representation of the piecewise linear functions. The most efficient representations are those adhering to the logarithmic principle in which the number of binary variables required to represent a piecewise linear function is bounded by above by $O(\log(n))$, where n is the number of pieces. Recent general-purpose solvers like CPLEX and Gurobi can now handle piecewise linear functions. In our nonlinear solver we make use of Gurobi's implementation of piecewise linear functions to solve MILPs with piecewise linear objectives.

The initialization method from Section 4.2 sets up a starting point for our iterative mechanism and the solution of the resulting modified MILP described in the paragraph above will provide a solution $\mathtt{x}^*$ that is feasible w.r.t. the set of constraints of the problem and that lies within a relative tolerance smaller than ϵ_0. Please recall that our objective is to approximate problem (1) within a tolerance of $\epsilon \leq \epsilon_0$. Moreover, because the point found $\mathtt{x}^*$ is feasible w.r.t. the set of constraints, evaluating $\mathtt{x}^*$ on the functions $f_i, i = 1 \dots n$ provides a valid upper bound for the problem. If the optimal value of the modified MILP (that provides a lower bound of the problem) happens to be off by at most ϵ from the upper bound, the method ends and returns $\mathtt{x}^*$. Otherwise we apply the objective-driven refinement

described next.

4.4 Objective-driven refinement

In the case where the lower and upper bounds are off by more than ϵ in relative terms, a repair procedure is invoked with the objective of tightening the current piecewise linear relaxation. The proposed tightening mechanism possesses a very distinctive feature when compared to similar —iterative refinement— methods. It explicitly aims at increasing the precision of the piecewise linear relaxation in the intervals of interest. This feature is what makes our method converge in a finite number of steps (see the Proposition 1), and to require in practice a low number of iterations to converge. This feature —which lies at the core of the proposed algorithm— is not present in previous iterative solvers such as those of Burlacu et al. (2020); Burlacu (2021); Nagarajan et al. (2019), where the intervals are divided according to their size, and the increased precision achieved only as a result of making the intervals smaller each time.

Let $\kappa \geq 1$ denote the number of times that the modified MILP has been solved. Let $\mathtt{x}_i^*$ be the i-th component of the solution $\mathtt{x}^*$ to the modified MILP. Let $\Delta_i^{\mathtt{x}^*}$ be the set of indices to the linear pieces containing the point $\mathtt{x}_i^*$. Let $\Delta_i \supseteq \Delta_i^{\mathtt{x}^*}$ be a set of indices to contiguous linear pieces and covering an interval $[l(\Delta_i), u(\Delta_i)]$. If $f_i(\mathtt{x}_i^*) - \alpha_i^k \mathtt{x}_i^* - \beta_i^k \leq \epsilon |f_i(\mathtt{x}_i^*)|$ for every $k \in \Delta_i^{\mathtt{x}^*}$, then the relaxation of f_i at the point $\mathtt{x}_i^*$ is sufficiently tight. If not, it needs to be tightened. We propose two objective-driven methods to tighten the linear pieces in Δ_i of the relaxation of f_i. They are referred to as the *conservative* and the *aggressive tightening*, and differ in the speed at which convergence is achieved. In each case, the new tolerance for the pieces in Δ_i is set according to one of the two equations below:

Conservative tightening

$$\epsilon' \leftarrow \frac{1}{2} \min \left\{ \epsilon(l^k) : k \in \Delta_i \right\}. \tag{2}$$

Aggressive tightening

$$\epsilon' \leftarrow \frac{\epsilon_0}{2^\kappa}. \tag{3}$$

By finding a piecewise linear relaxation for f_i using ϵ' as a tolerance within the interval $[l(\Delta_i), u(\Delta_i)]$ we are indeed tightening the relaxation of problem (1). This procedure will result in replacing one or a few linear pieces by multiple other pieces computed optimally using `LinA.jl`. However, the linear pieces surrounding those indexed by Δ_i will remain unchanged, and the scope of the update procedure will remain local. Moreover, in our method we make sure that `LinA.jl` is applied only on intervals $[l(\Delta_i), u(\Delta_i)]$ that verify $u(\Delta_i) - l(\Delta_i) \geq \delta$. If a Δ_i with $u(\Delta_i) - l(\Delta_i) < \delta$ needs to be tightened, then Δ_i is augmented with neighboring contiguous linear pieces until the interval size of at least δ is reached, before `LinA.jl` is applied. Parameter δ is therefore not a stopping criterion of our method, but a parameter that prevents our method from having to refine intervals that may be arbitrarily small.

It is easy to see that the ϵ' values constructed by applying the aggressive tightening rule are smaller than or equal to those that can be achieved if one applies instead the conservative tightening rule. One can expect that albeit being equally fast in the worst case, the latter shall provide quicker convergence in practice.

Once the repair procedure has been applied to every function f_i, the resulting MILP is solved again and the procedure repeated. The process ends when the lower and upper bounds are within a tolerance of ϵ. The following proposition provides a worst-case guarantee of convergence of the method to the desired tolerance.

4.5 Convergence of the method

We now provide two theoretical results that set the foundations of our method's efficiency. We must assume that the number of times that the functions f_i change in concavity is finite. This is a

requirement for `LinA.jl` to work and converge. In practice, this is a mild requirement as most cost functions interesting in practice satisfy it.

We will now prove that our method ends within a finite number of iterations of the main loop at Algorithm 1, and provides a feasible solution $\mathtt{x}^*$ whose associated objective is at most ϵ off the optimal solution.

Proposition 1. *If the domain of each variable x_i is bounded within the interval $[l_i, u_i]$ with $l_i \leq u_i$, the algorithm ends in at most*

$$N(\epsilon) = \left\lceil \log_2 \left(\frac{\epsilon_0}{\epsilon} \right) \right\rceil \sum_{i=1}^{n} \left\lceil \frac{u_i - l_i}{\delta} \right\rceil \tag{4}$$

iterations of the main loop at Algorithm 1 and provides a solution $\mathtt{x}^*$ that is far from the optimum by at most ϵ.

Proof. At every iteration, the method either finds a solution $\mathtt{x}^*$ that is at most ϵ off the optimal, or detects one set Δ_i of contiguous linear pieces of size $\geq \delta$ (possibly after having to augment $\Delta_i^{\mathtt{x}^*}$ with neighboring pieces) including all pieces containing $\mathtt{x}_i^*$ to apply the refinement. When applying equation (2) or (3), the tolerance associated with the pieces in Δ_i in the new relaxation will be cut of at least half. The number of times that this can happen before a linear piece within a given region is tightened two consecutive times is bounded above by $\lceil (u_i - l_i)/\delta \rceil$, and the number of times that this can happen before reaching the desired tolerance is bounded above by $\lceil \log_2(\epsilon_0/\epsilon) \rceil$. Because at each iteration there is at least one tightening for at least one variable $\mathtt{x}_i^*$, the sum of these quantities along the n dimensions of the domain is an upper bound for the total number of iterations before reaching global convergence. □

We now provide a comparative worst-case analysis between the proposed method and that of Burlacu et al. (2020); Burlacu (2021), which is also an adaptive-partitioning-based iterative method with finite convergence as proven by the authors. We will show that for Lipschitz-continuous functions f_i (i.e. such that $\exists L_i > 0, \forall x, y \in [l_i, u_i], |f_i(x) - f_i(y)| \leq L_i|x - y|$) the maximum number of main loops of Algorithm 1 required by our method to converge within a relative tolerance of ϵ is bounded above by $O(\log(N_B(\epsilon)))$, where $N_B(\epsilon)$ is the number of iterations in the worst-case required by Burlacu (2021)'s method to achieve the same precision.

Proposition 2. *Let us assume that the functions f_i are Lipschitz-continuous with constants $L_i, i = 1 \ldots n$. Then, for every $\epsilon > 0$, Burlacu's converges within a relative tolerance of ϵ in at most*

$$N_B(\epsilon) = 2^q \tag{5}$$

iterations, with $q = \left\lceil \log \left(\frac{\rho L}{\epsilon} \right) \right\rceil, \rho = \max\{u_i - l_i : i = 1 \ldots n\}, L = \max\{L_i : i = 1 \ldots n\}$.

Proof. See Burlacu et al. (2020). □

Proposition 3. *Let us assume that the functions f_i are Lipschitz-continuous with constants $L_i, i = 1 \ldots n$ and that $\epsilon_0 > 0$ is fixed. Then for every $\epsilon > 0$, $N(\epsilon) \leq O(\log(N_B(\epsilon)))$.*

Proof. Let us denote $\alpha = \sum\{\lceil (u_i - l_i)/\delta \rceil : i = 1 \dots n\}$. Starting from equation (5) we have that

$$\begin{aligned}
\log(N_B(\epsilon)) = q = \left\lceil \log\left(\frac{\rho L}{\epsilon}\right)\right\rceil = \left\lceil \log\left(\frac{\rho L}{\epsilon}\cdot\frac{\epsilon_0}{\epsilon_0}\right)\right\rceil &= \left\lceil \log\left(\frac{\rho L}{\epsilon_0}\cdot\frac{\epsilon_0}{\epsilon}\right)\right\rceil \\
&\geq \left\lceil \log\left(\frac{\rho L}{\epsilon_0}\right)\right\rceil + \left\lceil \log\left(\frac{\epsilon_0}{\epsilon}\right)\right\rceil - 1 \\
&\geq \frac{1}{2}\left\lceil \log\left(\frac{\epsilon_0}{\epsilon}\right)\right\rceil \\
&= \frac{1}{2}\left\lceil \log\left(\frac{\epsilon_0}{\epsilon}\right)\right\rceil \cdot \frac{\alpha}{\alpha} \\
&= \frac{N(\epsilon)}{2\alpha},
\end{aligned}$$

which implies that $N(\epsilon) \leq 2\alpha \log(N_B(\epsilon)) = O(\log(N_B(\epsilon)))$. □

4.6 Illustrative example

We will now illustrate our method by means of a very simple example. Let us consider the following fixed-charge nonlinear problem on two continuous variables:

$$\min \quad f_1(x_1) + f_2(x_2) \tag{6}$$

subject to

$$2x_1 + x_2 \geq 1 \tag{7}$$
$$2x_1 + 5x_2 \leq 4 \tag{8}$$
$$5x_2 \geq 2 \tag{9}$$
$$0 \leq x_1, x_2 \leq 1, \tag{10}$$

where

$$\begin{aligned}
f_1(x) &= \sin(4\pi x) - 0.4x + (2.4x)^2 + 5 \\
f_2(x) &= \sin(4\pi x) - 0.4x + (2.9x)^2 + 4.
\end{aligned}$$

In Figure 3 we depict the progression of our method for three iterations. We assume that the parameters ϵ_0, δ are defined as $\epsilon_0 = 0.2, \delta = 0.001$, although our method in these three iterations would be indifferent to any value of $\delta < 0.01$. Figures (3a)-(3b) depict the relaxation of the functions f_1, f_2 to an 80% of precision (i.e. by making $\epsilon_0 = 0.2$). This relaxation consists of two linear pieces for f_1, and of three for f_2. The optimal values taken by both variables in the resulting MILP are also depicted in these figures, and the quality of the relaxation given by the difference between the point evaluated in the red line (the original function f_i) and the blue line (the relaxation). For the second iteration, only function f_1 needs to be tightened, in the interval $[0, 1]$ as the optimal point x_1^* in this case lies at the intersection of the two linear pieces. We use a decreased tolerance of $\epsilon = 10$. The refinement leads to five linear pieces for the relaxation of f_1. We see that the variable x_1^* only slightly moves to a different region in the optimal solution of the resulting MILP as we observe in Figures (3c)-(3d). For the third iteration —depicted in Figures (3e)-(3f)—, we have again to tighten the relaxation of f_1 in the interval $[0.2, 0.7]$. Again, this tightening results in an increase in the number of linear pieces for the relaxation of f_1 to a total of seven. The optimal solution of the relaxation is attained at the point $(0.36, 0.4)$. Across these three iterations, the lower bound has changed from 8.713 in the first iteration to 8.577 in the second to 8.615 in the third. Meanwhile, the upper bound is of 8.862, which gives a 1.78% of optimality gap. For comparison, Codsi et al. (2021)'s method would

construct 10 linear pieces for each function, to achieve the same solution quality, a net increase of ten linear pieces with respect to the largest piecewise linear relaxation constructed using our iterative approach. As we will show later in Section 5, this difference increases significantly as we go to finer tolerances, with the linear relaxations constructed using our iterative method being often smaller by orders of magnitude.

Several remarks are in order. First, we observe that the dual bounds are not necessarily monotone across the method. This is due to the fact that a linear relaxation may not be tighter in the whole interval after a refinement. Second, our algorithm works by detecting the promising zones for successive refinements, while ignoring the parts that obviously may never be part of an optimal solution. This is what contributes at keeping the MILPs tractable along the whole solution process. The irrelevant zones of the domain, which can be seen as *noise*, should be approximated as roughly as possible, to focus one's efforts into refining the zones that are most likely to contain an optimal solution of the problem. Third, our method does not require the solution of a NLP oracle to ensure convergence —at the expense of being restricted to a polyhedral domain—, as it relies purely on the solution of mixed-integer linear programs. Fourth, our method computes lower and upper bounds at each iteration. Fifth, the refinement procedure is based on the computation of the best piecewise linear underestimation on a given interval for univariate functions for a predefined tolerance, instead of an arbitrary split of the interval into two or three. Sixth, our method can handle relative or absolute tolerances. As shown by Ngueveu (2019), the former may produce smaller MILPs for the same value of ϵ. Seventh, we do not require the piecewise linearizations to be continuous nor to interpolate the nonlinear functions, which can be efficiently exploited for instance by using the method of Codsi et al. (2021) to compute and tighten the relaxations.

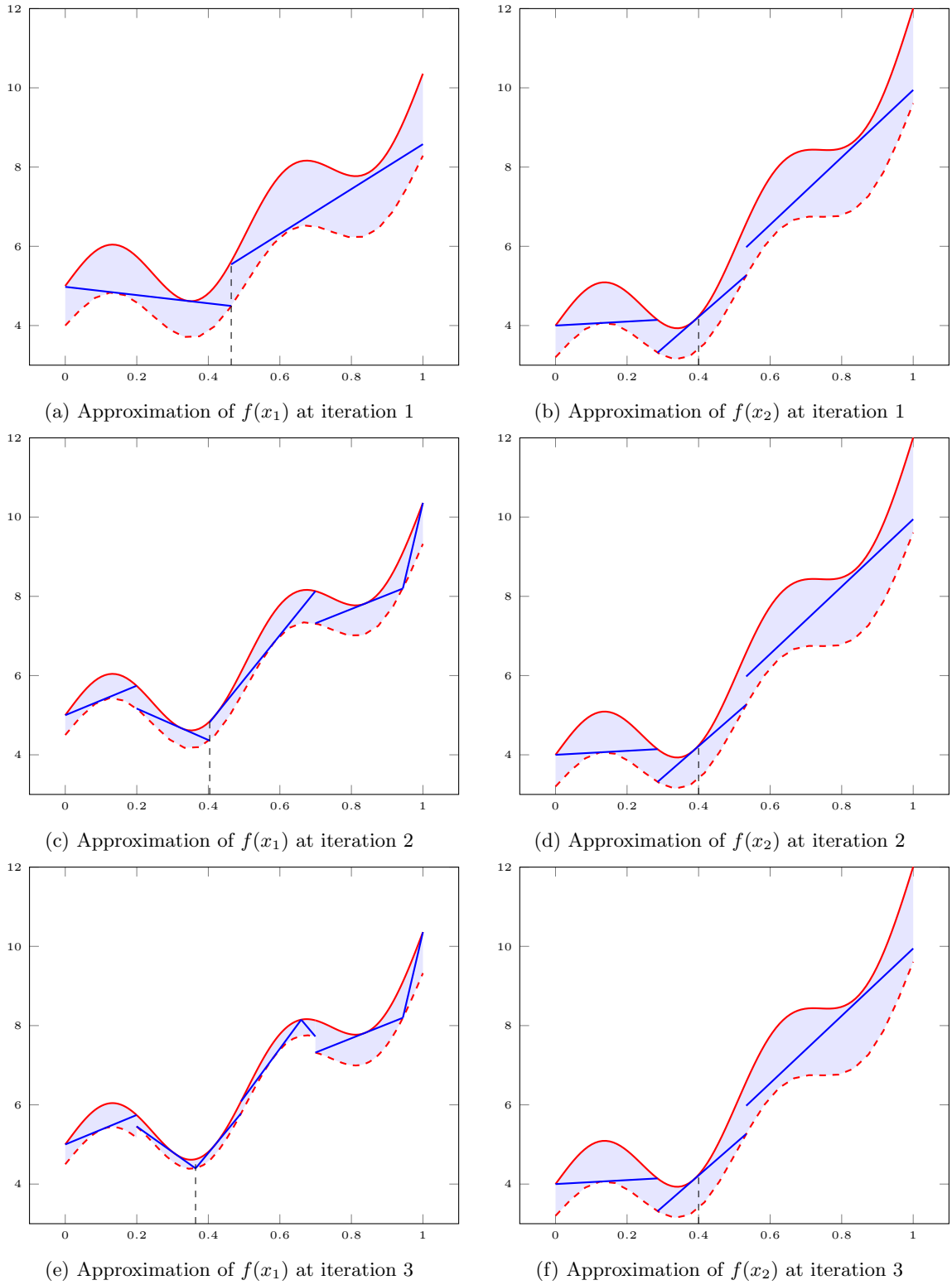
5 Applications

In this section we describe the application of our method to five classes of mixed-integer nonlinear optimization problems, namely: the transportation problem (TP), the uncapacitated facility location problem (UFLP), the multi-commodity flow problem (MCF), the multi-commodity network design problem (MCNDP), and the continuous knapsack problem (CKP). These problems differ structurally in the types and number of decision variables and constraints that are required to address them, and also at the level at which non-linearities are expressed. In knapsack or transportation problems the non-linearities are usually associated with all the structural decision variables. Flow or network design problems are often faced to non-linearities at one set of variables out of several. When that happens, non-linearities may or may not be dominant with respect to the linear terms in the objective, which in turn results in problems with varying degrees of difficulty. Our selection of problems, and of levels at which non-linearities are expressed, intend to provide a comprehensive sample of problem variants with varying degrees of difficulty.

First, we introduce 10 classes of nonlinear functions that we will use to assess the performance of our method. Next, we describe the experimental setup used throughout our campaign. Then, we present nonlinear variants of the five optimization problems considered in our computational study and provide computational evidence of the performance of our approach to reach near-optimal solutions with very tight numerical guarantees.

5.1 Nonlinear cost functions

In our tests we consider ten univariate functions defined in the interval $[0, 1]$ and such that $f(1) = 1$ or 2. We will later explain how we adapt these functions to situations where the domain and values



(a) Approximation of $f(x_1)$ at iteration 1

(b) Approximation of $f(x_2)$ at iteration 1

(c) Approximation of $f(x_1)$ at iteration 2

(d) Approximation of $f(x_2)$ at iteration 2

(e) Approximation of $f(x_1)$ at iteration 3

(f) Approximation of $f(x_2)$ at iteration 3

Figure 3: Three iterations of the proposed method on problem (6)-(10)

are different. The functions are as follows:

$$f_1(x) = \begin{cases} 0.5 + \dfrac{1.5}{1 + \dfrac{e^{-10(x-0.1)}}{2}} & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$f_2(x) = \begin{cases} 2.5312499998168745x^3 - 2.812499999708163x^2 + 1.0312499999207962x + 0.25 & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$f_3(x) = 0.2743170648074066(\sin(2x) + x)^2$$

$$f_4(x) = 0.06123588766918159(\sin(5x) + 5x)^2$$

$$f_5(x) = 0.039137166375225234(\sin(10x) + 5x)^2$$

$$f_6(x) = 0.19399896533885155 \sin\left(\pi \frac{100x - 10}{40}\right) - 0.3879979306777031 \cos\left(\pi \frac{100x - 10}{80}\right) + 0.504397309881014$$

$$f_7(x) = \tfrac{1}{3} \log(1 + 19.085536923187668x)$$

$$f_8(x) = x^2$$

$$f_9(x) = \sqrt{x}$$

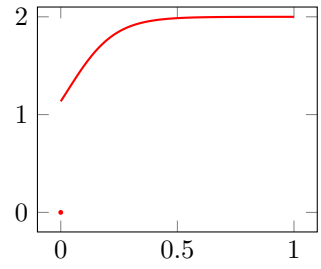
$$f_{10}(x) = \frac{1.0002776471474824}{1 + 44.742557e^{-15.2216(x-0.21229801)}}$$

These functions possess a variety of attributes. Functions f_1, f_2 contain a fixed-charge term that is only activated for strictly positive values of x. Functions f_7, f_9 are pure concave, while function f_8 is pure convex. Functions f_3, f_{10} have one inflection point, where the functions pass from convex to concave. Function f_5 is non-monotonic. Functions f_4, f_5, f_6 have two or more inflection points. All the functions are Lipschitz-continuous in the interval $]0, 1[$. Functions f_3, f_4, f_5, f_6 and f_{10} have been used in the past, for instance in Trindade et al. (2023), with the only modification being the application of a scaling coefficient to make sure that $f_i(1) = 1$. In Figure 4 we depict the ten cost functions in the interval $[0, 1]$.

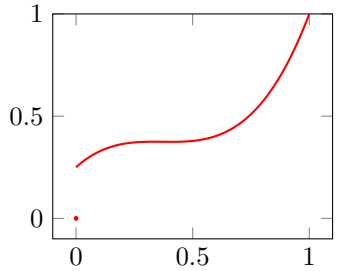
5.2 Experimental setup

We consider a target tolerance of $\epsilon = 10^{-4}$ because such value is sufficiently small to render the problems difficult and yet still large enough to prevent errors due to floating point arithmetic. Smaller values of ϵ resulted in a somewhat erratic behavior related to floating point errors in `LinA.jl` and `GUROBI`. We consider two variants of our method, namely one using the aggressive tightening rule, and a second one where we set $\epsilon_0 = \epsilon$. The former is denoted `CN24` and for this variant, we let $\epsilon_0 = 10\%$. The latter is denoted as `NAIVE`. For nonlinear functions valid in a domain $[l, u]$ we let $\delta = (u - l) \times 10^{-3}$. Our preliminary experience indicates that the difference between the conservative and the aggressive tightening rules is not important, with a slight edge for the aggressive over the conservative. For this reason, we omit in our analysis the variant of our method that considers the conservative tightening rule. In addition, we consider three other mixed-integer nonlinear solvers for benchmarking purposes (namely `SCIP`, `COUENNE` and `GUROBI`) at their default settings and using the same optimality tolerance of $\epsilon = 10^{-4}$. According to their documentation (Gurobi Optimization, LLC, 2024), `GUROBI`'s MINLP solver combines spatial branch-and-bound with piecewise linear approximations to handle the nonlinearities. That makes the comparison against our solver particularly interesting as both solvers (`GUROBI` and `CN24`) rely at least up to some extent in approximating the nonlinearities using piecewise linear terms. A fourth method (`Alpine.jl`, Nagarajan et al. (2016)) has been initially given consideration but finally dropped from our main computational campaign as its performance failed

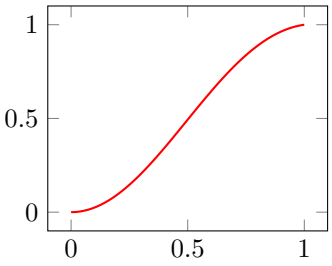
Figure 4: Depictions of ten types of cost functions



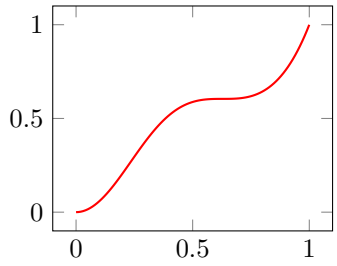
(a) Functions of the type f_1



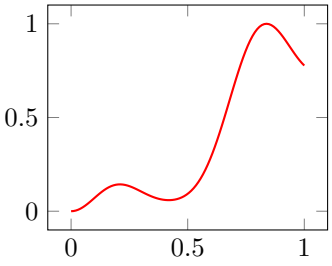
(b) Functions of the type f_2



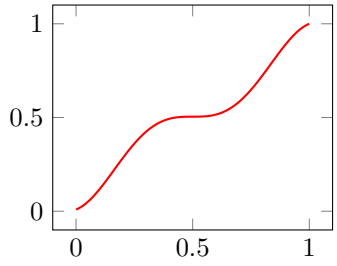
(c) Functions of the type f_3



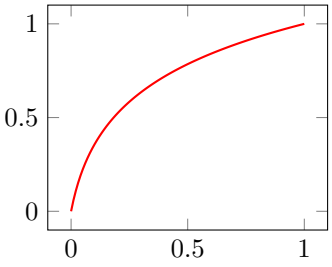
(d) Functions of the type f_4



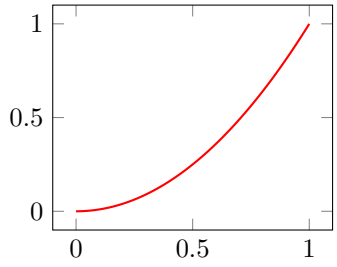
(e) Functions of the type f_5



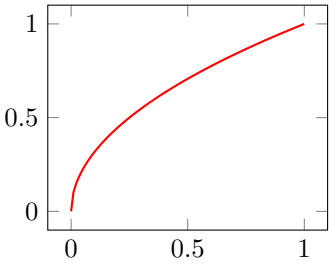
(f) Functions of the type f_6



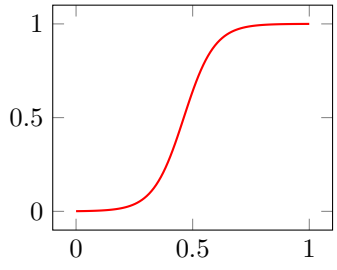
(g) Functions of the type f_7



(h) Functions of the type f_8



(i) Functions of the type f_9



(j) Functions of the type f_{10}

to match that of all the other methods by a large extent. We provide a summary analysis of that method in Appendix A. A summary description of the algorithms used in our benchmarking can be found in Table 1.

Solver	Description	Reference
`CN24`	Our method with the aggressive tightening rule	This paper
`NAIVE`	Our method with $\epsilon_0 = \epsilon$	This paper
`SCIP`	Direct solution of the MINLP using SCIP v8.0	Bestuzheva et al. (2023)
`COUENNE`	Direct solution of the MINLP using Couenne v0.5.8	Belotti et al. (2009)
`GUROBI`	Direct solution of the MINLP using Gurobi v12.0	Gurobi Optimization, LLC (2024)

Table 1: Five solvers for benchmarking purposes

Our method has been coded in Julia v1.10 using the JuMP 1.23 solver interface, and uses `GUROBI` 12.0 as general-purpose solver for the two variants of our method (`CN24` and `NAIVE`). The five solvers (our two plus `COUENNE`, `GUROBI` and `SCIP`) have been executed on an Intel(R) Xeon(R) Gold 6258R CPU @ 2.70GHz with 128 GB of RAM. We limit, however, the maximum resource consumption to 5GB of RAM. While the machine is capable of running code in parallel, for reproducibility purposes we limit the number of threads to one on all settings. In all cases we have set a time limit of 3,600 seconds before deeming a problem as unsolved. Please note that in `SCIP`, optimality tolerances are defined differently as for the other solvers. For given lower and upper bounds l, u of a problem, we consider (and also `GUROBI`, and `COUENNE`) a relative gap defined as $(u-l)/\max\{|l|,|u|\} \times 100$. `SCIP`, on the other hand, considers the gap defined as $(u-l)/\min\{|l|,|u|\} \times 100$. Most of the time the differences will not be noticeable (a gap of say 1% in one definition may be mapped to a gap of say 0.99% with an alternative definition).

Our computational campaign comprises 5,980 different problem instances, considering the different problem variants, and cost functions. For that reason, in the main text we report aggregate results, and report detailed data in Appendix B. Note that in about 60 problems out of the 5,980 we observed discrepancies in the results of the different methods, possibly due to floating-point arithmetic errors in the different solvers. The tests for which discrepancies were observed are omitted from the analyses, and are marked in red in Appendix B.

In our analysis we report the shifted geometric means of the computing times for each algorithm and the number of problems approximated successfully within a tolerance of $\epsilon = 10^{-4}$. The shifted geometric mean of a collection of values $(v_i)_{i=1}^N$ and a shift value of $\sigma \geq 0$ is computed as

$$\sqrt[N]{\prod_{i=1}^{N}(v_i+\sigma)} \quad - \quad \sigma. \tag{11}$$

The shifted geometric mean has some nice properties that have made it the standard metric for benchmarking optimization software: it is less sensitive to large outliers than the arithmetic mean, and also less sensitive to small outliers than the geometric mean. We use a shift value of $\sigma = 10$, which is also the value used in Burlacu (2021) and in H. Mittelmann's benchmarks `https://plato.asu.edu/bench.html`. For the purpose of computing the shifted geometric means, time limits are given a value of four hours, or equivalently 14,400 seconds. This is four times the time limit allowed.

5.3 Transportation problem

In the transportation problem (TP, Ford Jr and Fulkerson, 1956), we are given a set U of n origins, and a set V of m destinations. With each origin $u \in U$ we associate an offer $o_u > 0$ of a given commodity, and with each destination $v \in V$ a demand $d_v > 0$ of the same commodity and such

that $\sum_{u\in U} o_u = \sum_{v\in V} d_v$. For the sake of simplicity, we assume that the offers and demands are all integer-valued. For each pair $(u, v) \in U \times V$, we are given a cost function $g_{uv} : \mathbb{R} \to \mathbb{R}$ such that $g_{uv}(x)$ represents the cost of transporting x units of flow from u to v. The objective is to select the amounts $(x_{uv})_{u\in U,v\in V}$ to transport along the arcs $(u, v) \in U \times V$ such that: 1) each origin $u \in U$ sends exactly o_u units of flow; 2) each destination $v \in V$ receives exactly d_v units of flow; and 3) the total cost $z = \sum_{(u,v)\in U\times V} g_{uv}(x_{uv})$ is minimized.

If the cost functions are all linear, the TP described above resorts to a classical linear TP, which can be solved in polynomial time using a network flow algorithm (Kleinschmidt and Schannath, 1995). In this article, we are interested in the scenario where the cost functions are separable but otherwise might take arbitrary forms (for instance non-convex). The problem can be modeled as a non-linear optimization problem using the notation already introduced as follows:

$$\texttt{minimize} \quad z = \sum_{u\in U, v\in V} g_{uv}(x_{uv}) \tag{12}$$

subject to

$$\sum_{v\in V} x_{uv} \le o_u \qquad u \in U \tag{13}$$

$$\sum_{u\in U} x_{uv} \ge d_v \qquad v \in V \tag{14}$$

$$x_{uv} \ge 0 \qquad u \in U, v \in V. \tag{15}$$

To assess the effectiveness of our approach to approximate the nonlinear TP (NLTP), we have generated a set of random instances, as follows. We consider squared problems with $n = m \in \{5, 10, 15, 20\}$ origins and destinations in an Euclidean space on a square of dimensions 100×100. We consider nonlinear cost functions $g_{uv}(x_{uv}) = c_{uv} f_i(x_{uv}/\xi_{uv})$ for each possible cost function f_i found in Section 5.1, where c_{uv} is a nominal cost for using an arc (u, v) —computed as the Euclidean distance between the points plus a random noise added from an uniform distribution in the interval $[-3, 3]$— and where $\xi_{uv} = \min\{o_u, d_v\}$.

Next, we compare `CN24` against `COUENNE`, `SCIP`, `GUROBI` and `NAIVE`, and present aggregate computational results in Table 2. We report, for each solver and for each cost function, the total number of problems solved (#S), the shifted geometric means of the computing times (CPU) and the shifted geometric means of the gaps (GAP). The summary row *Total* serves at reporting the total number of problems approximated to the desired tolerance.

		Gurobi			**Couenne**			**SCIP**			**Naive**			**CN24**		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_1	40	4	8114.7	31.2	3	10816.6	50.8	9	5980.9	30.3	11	4852.8	27.4	**19**	**1643.2**	**1.2**
f_2	40	**28**	**389.9**	**0.4**	0	14400.0	100.0	0	14400.0	100.0	10	6313.5	30.9	17	3301.9	1.8
f_3	40	5	9023.6	12.2	0	14400.0	37.8	1	13570.5	47.1	11	6876.1	27.4	**17**	**3307.3**	**0.0**
f_4	40	10	5105.9	17.2	0	14400.0	29.9	7	8474.2	24.9	12	6107.5	24.3	**16**	**3185.9**	**0.9**
f_5	40	9	6131.3	**18.8**	0	14400.0	31.4	0	14400.0	51.7	7	10518.4	44.0	**10**	5380.0	19.0
f_6	40	4	10272.6	47.3	0	14400.0	91.8	1	13872.0	89.0	0	14400.0	100.0	**12**	4618.0	**1.8**
f_7	40	**32**	**106.2**	**0.6**	11	2543.0	23.1	30	263.4	1.0	0	14400.0	100.0	19	1459.5	2.5
f_8	39	**39**	**0.2**	**0.0**	19	592.5	0.3	**39**	4.7	**0.0**	**39**	194.0	**0.0**	27	487.0	**0.0**
f_9	40	**31**	**137.1**	**0.7**	10	2927.0	9.9	16	1208.2	8.0	0	14400.0	100.0	17	1794.7	3.5
f_{10}	40	0	14400.0	90.1	0	14400.0	97.9	0	14400.0	99.6	19	3939.1	10.3	**20**	**1485.9**	**0.9**
Total	399	162			43			103			109			**174**		

Table 2: Aggregate results for problem NLTP

The results show that our method is competitive against `GUROBI`, with both methods being able to solve a comparable number of problems to proven optimality. Both methods, on the other hand, show a significant edge over `SCIP` and `COUENNE`. We also observe that `CN24` outperforms the `NAIVE` variant of the method, allowing to solve about 50% more problems. We also observe that the proposed method

performs best when the objective is non-convex, as it does not exploit convexity in any specific way, as opposed to the other methods that behave particularly well on the convex objectives f_8.

5.4 Uncapacitated facility location

The uncapacitated facility location problem (UFLP, Cornuéjols and Thizy, 1982) deals with the problem of deciding the location of one or more facilities among an universe U of n total facilities, and to assign m customers in a set V to the selected facilities. We associate an opening cost $\gamma_u > 0$ to each facility $u \in U$, and an assignment cost $c_{uv} > 0$ to each assignment of a customer v to a facility u. Each customer $v \in V$ has a demand of $d_v > 0$ units. We seek to determine: 1) what facilities to open; 2) what fraction of the demand of a customer must be assigned to each open facility; 3) at minimum total cost. The non-linearities in this problem may come from two sources:

Nonlinear warehousing costs (NLUFLP-W) obtained by replacing the opening cost γ_u of the facilities by a nonlinear term $g_u(\cdot)$ representing the warehousing cost associated with the service of the demand fulfilled by facility u. The NLUFLP-W arises for instance in problems with congestion (Harkness and ReVelle, 2003) or with economies/diseconomies of scale (Lu et al., 2014).

Nonlinear assignment costs (NLUFLP-A) obtained by replacing each assignment cost c_{uv} by a nonlinear assignment function $h_{uv}(\cdot)$ such that $h_{uv}(s)$ represents the cost associated with servicing a fraction $s \in [0, 1]$ of the demand of customer v by facility u. Such nonlinear cost function models the effects of spatial interaction in facility location models (Holmberg, 1999).

Let us introduce a model that includes both settings at once. Let s_u be a continuous variable representing the amount of demand serviced by facility u. For each possible assignment (u, v) of a customer v to a facility u, let x_{uv} be the fraction of the demand of v that is serviced by facility u. Using the notation already introduced, the following model solves the UFLP while minimizing the total nonlinear costs:

$$\texttt{minimize} \quad z = \sum_{u \in U} g_u(s_u) + \sum_{u \in U, v \in V} h_{uv}(x_{uv}) \tag{16}$$

subject to

$$\sum_{u \in U} x_{uv} = 1 \qquad v \in V \tag{17}$$

$$\sum_{v \in V} d_v x_{uv} - s_u = 0 \qquad u \in U \tag{18}$$

$$x_{uv} \geq 0 \qquad u \in U, v \in V. \tag{19}$$

Note that this model does not include binary variables for the opening of the facilities. Indeed, we assume that the fixed costs, if any, can be included in the cost functions $g_u(s_u)$, for instance in the form of a fixed-charge cost term like those appearing in the functions f_1, f_2 considered in our computational study.

We consider the two possible scenarios for the source of the non-linearities separately: a first scenario with nonlinear warehousing costs, but with linear assignment costs; and a second scenario with nonlinear assignment costs, but with constant fixed costs. In both cases, we consider the ten classes of nonlinear cost functions defined in Section 5.1. For a given class f_i, we consider the nonlinear warehousing cost $g_u(s_u)$ equal to $\gamma_u f_i(s_u/D)$, where $D = \sum\{d_v : v \in V\}$. The nonlinear assignment cost, on the other hand, is computed as $h_{uv}(x_{uv}) = c_{uv} f_i(x_{uv})$. Note that when considering the variant with nonlinear assignment costs only, we assume that $g_u(s_u) = \gamma_u$ if $s_u > 0$, 0 otherwise.

We consider three sets from the facility location literature:

- The ORLib dataset introduced by Beasley (1990). This benchmark dataset contains 40 problems with a number of facilities and customers ranging between $[16, 100]$ and $[50, 1000]$. The three largest problems in this dataset (each comprising 100 potential facilities and 1,000 customer nodes) proved to difficult for all the methods. Hence, we restrict our analysis to the 37 smaller instances, with $|U|$ ranging in $[16, 50]$ and $|V| = 50$.
- The Holmberg dataset introduced in Holmberg et al. (1999). It consists of 71 instances with the number of facilities ranging between 10 and 30, and the number of customer nodes ranging between 50 and 200.
- A set of randomly generated instances according to the descriptions provided in Günlük and Linderoth (2010) (hereafter referred to as the Günlük dataset), with $|U|$ ranging between 10 and 50 and such that $|V| = 2|U|$.

For the variant with nonlinear warehousing costs, the results reported in Table 3 indicate that our method shows a slight edge over all the other methods. It is able to successfully approximate all but two problems to proven optimality, out of 1559 in total, in usually much shorter computing times that all the other methods. While `GUROBI` was often faster, it shows a lesser degree of robustness from the results obtained for the cost functions f_2, f_7, and f_9. It is remarkable that `GUROBI`, which only very recently included support for nonconvex MINLPs, shows the best performance among all the other solvers.

		Gurobi			**Couenne**			**SCIP**			**Naive**			**CN24**		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_1	157	**157**	**0.5**	**0.0**	**157**	54.0	**0.0**	151	41.6	0.2	156	70.2	0.0	**157**	5.3	**0.0**
f_2	143	103	126.7	1.7	98	592.9	2.1	60	1649.3	6.0	**143**	575.9	**0.0**	**143**	**23.8**	**0.0**
f_3	157	**157**	**0.6**	**0.0**	99	338.5	0.2	**157**	6.7	**0.0**	**157**	218.2	**0.0**	**157**	15.3	**0.0**
f_4	158	**158**	**0.8**	**0.0**	85	627.1	2.3	**158**	9.9	**0.0**	**158**	313.9	**0.0**	**158**	52.6	**0.0**
f_5	158	**158**	**4.1**	**0.0**	91	541.2	2.4	155	33.2	0.1	157	876.1	0.0	**158**	55.1	**0.0**
f_6	158	**158**	**2.4**	**0.0**	117	287.3	0.2	27	5113.4	22.5	50	4159.0	22.4	**158**	91.2	**0.0**
f_7	158	109	348.8	1.4	25	6230.4	6.0	0	14400.0	58.5	32	9062.7	38.7	**156**	**21.5**	**0.1**
f_8	157	**157**	**0.2**	**0.0**	**157**	11.7	**0.0**	**157**	25.6	**0.0**	**157**	37.8	**0.0**	**157**	3.0	**0.0**
f_9	155	73	1156.4	3.3	10	11397.4	16.5	0	14400.0	48.3	1	14268.6	97.0	**155**	**26.2**	**0.0**
f_{10}	158	**158**	**1.7**	**0.0**	**158**	37.5	**0.0**	116	229.6	1.2	**158**	150.5	**0.0**	**158**	14.9	**0.0**
Total	1559	1388			997			981			1169			**1557**		

Table 3: Aggregate results for problem NLUFLP-W

For the variant with nonlinear assignment costs, the aggregate results reported in Table 4 indicate that it is now `GUROBI` which shows a slight edge over all other solvers, with `SCIP` coming in a close second place, and `CN24` in third place to complete the podium. The `NAIVE` variant of our method, which let us recall computes a single approximation of the cost functions to the desired tolerance, shows the poorest performance among all methods.

		Gurobi			**Couenne**			**SCIP**			**Naive**			**CN24**		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_1	158	**158**	**3.7**	**0.0**	11	10933.0	18.5	**158**	59.2	**0.0**	93	2920.4	5.7	**158**	604.0	**0.0**
f_2	158	**12**	9446.0	**2.4**	0	14400.0	100.0	0	14400.0	100.0	0	14400.0	100.0	9	12146.8	45.8
f_3	158	0	14400.0	51.7	0	14400.0	88.3	0	14400.0	60.7	0	14400.0	100.0	0	14400.0	**21.7**
f_4	158	0	14400.0	59.7	0	14400.0	88.1	0	14400.0	64.4	0	14400.0	100.0	**10**	11793.4	**25.0**
f_5	158	2	13743.9	**40.2**	0	14400.0	87.5	0	14400.0	58.7	0	14400.0	100.0	**3**	13701.6	80.4
f_6	158	0	14400.0	96.2	0	14400.0	99.8	0	14400.0	97.1	4	13801.3	88.9	**11**	11862.5	**23.1**
f_7	158	**158**	**0.4**	**0.0**	**158**	68.3	**0.0**	**158**	7.0	**0.0**	0	14400.0	100.0	149	355.2	0.0
f_8	155	**146**	**45.2**	**0.1**	0	14400.0	63.4	140	161.3	0.3	6	13365.2	83.5	14	10703.7	5.1
f_9	158	**158**	**1.2**	**0.0**	122	711.9	1.9	156	27.5	0.0	0	14400.0	100.0	142	567.8	0.2
f_{10}	158	0	14400.0	88.8	0	14400.0	97.9	0	14400.0	94.8	0	14400.0	100.0	**10**	11649.6	**3.4**
Total	1577	**634**			291			612			103			506		

Table 4: Aggregate results for problem NLUFLP-A

5.5 Multi-commodity flow

The multicommodity flow problem (MCFP, Assad, 1978) deals with the dispatching of multiple distinguishable commodities throughout a network. We are given a set of nodes N, a set of arcs A and a set of commodities P. Each node $i \in N$ has a maximal capacity Q_i, and a demand of commodity p denoted D_i^p. Each arc $(i,j) \in A$ has a capacity U_{ij} for the maximum amount of flow that can pass through it. With each commodity $p \in P$ we associate a quantity to be shipped W^p and a unit routing cost function over arc (i,j) denoted $R_{ij}^p(\cdot)$. In problems with congestion, we typically observe a nonlinear cost $g_i(v_i)$ incurred on each node $i \in N$ for the amount of flow traversing the node, where v_i denotes that flow.

The problem can be posed as a NLP, as follows. For every arc $(i,j) \in A$ and commodity $p \in P$ we let x_{ij}^p be the amount of flow of commodity p traversing the arc. For every node $i \in N$ we let v_i be the amount of flow entering node i. We consider the following NLP:

$$\texttt{minimize} \quad z = \sum_{(i,j)\in A} \sum_{p\in P} R_{ij}^p(x_{ij}^p) + \sum_{i\in N} g_i(v_i) \tag{20}$$

subject to

$$\sum_{j\in N_i^+} x_{ij}^p - \sum_{j\in N_i^-} x_{ji}^p = D_i^p, \qquad i \in N, p \in P \tag{21}$$

$$\sum_{p\in P} x_{ij}^p \le U_{ij}, \qquad (i,j) \in A \tag{22}$$

$$\sum_{j\in N:(j,i)\in A} \sum_{p\in P} x_{ji}^p - v_i = 0, \qquad i \in N \tag{23}$$

$$0 \le x_{ij}^p \le W^p, \qquad (i,j) \in A, p \in P \tag{24}$$

$$0 \le v_i \le Q_i, \qquad i \in N. \tag{25}$$

We consider two variants of this problem for our computational analysis, described as follows:

Nonlinear routing costs, no congestion at the nodes (NLMCFP-A). This variant corresponds to ignoring the terms $g_i(v_i), i \in N$ in the objective. Note that in this case, it is possible to get rid of the variables v_i in the model.

Nonlinear congestion costs, no routing costs (NLMCFP-N). This variant corresponds to ignoring the terms $R_{ij}^p(x_{ij}^p)$ in the objective.

Our experimental campaign considers some classical datasets from the literature, namely the so-called C and C+ instances for the multicommodity network design problem (Crainic et al., 2001). These datasets comprise a total of 43 problem instances and contain between 10 and 30 nodes and between 10 and 200 commodities. To construct the nonlinear cost functions $g_i(\cdot)$, we consider the fixed costs γ_i defined in the instance files, and then, for every cost function $f_l, l = 1 \dots 10$ we do $g_i(v_i) = \gamma_i f_l(v_i/Q_i)$ defined in the interval $[0, Q_i]$ for every node. For the routing cost functions R_{ij}^p we use the nominal costs C_{ij}^p and let $R_{ij}^p(x_{ij}^p) = C_{ij}^p f_l(x_{ij}^p/W^p)$, for each nonlinear function $f_l, l = 1 \dots, 10$ considered in our study.

For the first variant, the results reported in Table 5 show that, it is now the `NAIVE` method that provides the best overall performance, with `GUROBI` and `SCIP` coming in a close second and third place. Our method, along with `COUENNE`, provide the weakest performances.

For the second variant, the results reported in Table 6 show that it is now our method which performs best, solving significantly more problems than all its competitors. For the unsolved problems, moreover, the average gaps reported show a robust performance, with average gaps never exceeding 1.2%.

		Gurobi			Couenne			SCIP			Naive			CN24		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_1	42	0	14400.0	54.5	0	14400.0	95.4	0	14400.0	63.7	1	13824.5	89.5	**4**	10918.2	**37.9**
f_2	42	**1**	13131.9	**11.2**	0	14400.0	100.0	0	14400.0	93.1	0	14400.0	100.0	**1**	13766.7	38.9
f_3	43	0	14400.0	51.7	0	14400.0	88.0	0	14400.0	39.7	**11**	6351.3	19.8	0	14400.0	**3.4**
f_4	43	0	14400.0	66.4	0	14400.0	94.6	0	14400.0	59.9	**3**	12685.4	72.3	0	13943.3	**23.8**
f_5	39	0	14400.0	89.1	0	14400.0	99.3	0	14400.0	76.0	0	14400.0	100.0	0	14400.0	**71.4**
f_6	43	0	14400.0	79.2	0	14400.0	99.4	0	14400.0	84.1	0	14400.0	100.0	0	14400.0	**35.1**
f_7	42	**1**	13620.9	**25.5**	0	14400.0	82.8	0	14400.0	28.2	0	14400.0	100.0	0	14400.0	54.8
f_8	29	**29**	**5.5**	**0.0**	3	10289.6	61.1	**29**	251.3	**0.0**	20	978.0	0.8	2	11743.9	2.7
f_9	42	**1**	13408.1	**34.4**	0	14400.0	97.8	0	14400.0	41.0	0	14400.0	100.0	**1**	13663.1	50.5
f_{10}	43	0	14400.0	98.3	0	14400.0	99.8	0	14400.0	97.9	0	14400.0	100.0	**3**	12060.3	**7.1**
Total	408	32			3			29			**35**			11		

Table 5: Aggregate results for problem NLMCFP-A

		Gurobi			Couenne			SCIP			Naive			CN24		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_1	43	1	13338.9	2.6	0	14400.0	84.8	3	12296.8	4.4	**37**	**344.6**	0.9	36	475.9	**0.4**
f_2	43	9	6427.2	**0.3**	0	14400.0	100.0	0	14400.0	90.9	**36**	**1410.5**	1.1	28	2474.2	0.7
f_3	43	12	4394.1	6.9	0	14400.0	72.7	13	5054.7	1.5	**38**	**290.7**	0.5	29	1482.1	**0.1**
f_4	43	10	5621.2	10.0	0	14400.0	93.2	1	13550.3	38.3	**35**	**1121.2**	1.4	26	2933.8	**0.9**
f_5	43	9	5631.4	9.6	0	14400.0	82.0	11	6809.3	12.1	**37**	**736.3**	0.9	32	914.0	**0.4**
f_6	43	3	12641.4	11.1	0	14400.0	93.4	1	13709.2	33.2	**36**	**813.2**	**1.1**	26	3238.8	1.2
f_7	43	**34**	**330.1**	**0.5**	0	14400.0	62.2	19	2341.6	3.6	0	14400.0	100.0	16	3534.9	1.1
f_8	42	**42**	**4.0**	**0.0**	8	7917.7	30.9	**42**	43.5	**0.0**	41	47.8	0.0	37	357.6	0.0
f_9	43	**34**	**333.2**	**0.4**	0	14400.0	72.6	18	3924.6	4.2	0	14400.0	100.0	19	2942.2	0.6
f_{10}	43	0	14400.0	22.7	0	14400.0	97.5	0	14400.0	27.3	0	14400.0	100.0	**28**	**1045.6**	**0.1**
Total	429	154			8			108			260			**277**		

Table 6: Aggregate results for problem NLMCFP-N

5.6 Multi-commodity network design

The nonlinear multicommodity network design problem (NLMCND, Paraskevopoulos et al., 2016) is a generalization of the NLMCFP where a design variable y_{ij} for every arc $(i,j) \in A$ governs the decision of whether the arc can be used or not, for instance with the purpose of imposing fixed-charge costs $\gamma_{ij}, (i,j) \in A$. Using the same notation as before, in addition to the design variables $y_{ij}, (i,j) \in A$, the NLMCNDP can be posed as the following MINLP:

$$\texttt{minimize} \quad z = \sum_{(i,j)\in A} \sum_{p\in P} R_{ij}^p(x_{ij}^p) + \sum_{(i,j)\in A} \gamma_{ij} y_{ij} + \sum_{i\in N} g_i(v_i) \tag{26}$$

subject to

$$\sum_{j\in N_i^+} x_{ij}^p - \sum_{j\in N_i^-} x_{ji}^p = D_i^p, \qquad i \in N, p \in P \tag{27}$$

$$x_{ij}^p \le W^p y_{ij}, \qquad (i,j) \in A, p \in P \tag{28}$$

$$\sum_{p\in P} x_{ij}^p \le U_{ij} y_{ij}, \qquad (i,j) \in A \tag{29}$$

$$\sum_{j\in N:(j,i)\in A} \sum_{p\in P} x_{ji}^p - v_i = 0, \qquad i \in N \tag{30}$$

$$x_{ij}^p \ge 0, \qquad (i,j) \in A, p \in P \tag{31}$$

$$0 \le v_i \le Q_i, \qquad i \in N. \tag{32}$$

$$y_{ij} \in \{0,1\}, \qquad (i,j) \in A. \tag{33}$$

We again consider two variants of this problem, described as follows:

Nonlinear routing costs, no congestion at the nodes (NLMCNDP-A). This is achieved simply by ignoring the terms $g_i(v_i)$ in the objective.

Nonlinear congestion at the nodes, no routing costs (NLMCNDP-N). This corresponds to ignoring the terms $R^p_{ij}(x^p_{ij})$ in the objective.

We consider the same 430 problem instances as for the NLMCFP, but now with the explicit consideration of the design variables y_{ij} and fixed charge costs γ_{ij} for every arc $(i,j) \in A$. The nonlinear congestion costs $g_i(\cdot)$ and the nonlinear routing costs $R^p_{ij}(\cdot)$ are computed as in Section 5.5. The NLMCNDP is a more challenging problem than the NLMCFP due to the consideration of the fixed-charge costs.

From the results reported in Table 7 for the NLMCNDP-A we observe that our method shows a superior performance compared to all the others methods, being able to solve thrice as many problems when compared to `GUROBI`, the closest competitor. `SCIP` and especially `COUENNE` exhibit much weaker performances, being able to solve a small fraction of the entire testbed. Despite the superior performance of te proposed method when compared to all others, we still observe that the problem remains challenging, with a 95% of the testbed remaining out of reach for all solvers.

		Gurobi			**Couenne**			**SCIP**			**Naive**			**CN24**		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_1	43	0	14400.0	54.7	0	14400.0	91.8	0	14400.0	64.5	1	13781.6	89.7	**4**	11271.9	**43.0**
f_2	41	1	12763.4	**20.0**	0	14400.0	100.0	0	14400.0	98.8	0	14400.0	100.0	**2**	13039.2	49.1
f_3	42	0	14400.0	58.2	0	14400.0	89.1	0	14400.0	48.4	0	14400.0	100.0	**2**	12758.6	**27.5**
f_4	43	0	14400.0	70.4	0	14400.0	94.8	0	14400.0	66.9	0	14400.0	100.0	0	13941.9	**39.4**
f_5	43	0	14400.0	58.7	0	14400.0	93.2	0	14400.0	56.8	0	14400.0	100.0	**3**	12045.8	**42.5**
f_6	43	0	14400.0	62.3	0	14400.0	97.4	0	14400.0	79.3	0	14400.0	100.0	**2**	12290.7	**39.7**
f_7	41	**1**	13236.4	**24.1**	0	14400.0	91.8	0	14400.0	47.3	0	14400.0	100.0	**1**	12449.4	69.8
f_8	39	**4**	7488.5	26.5	2	13258.3	71.0	**4**	7657.8	**20.9**	2	11283.4	70.3	3	11144.2	39.2
f_9	38	**1**	13213.7	**22.3**	**1**	11905.1	88.4	0	14400.0	45.4	0	14400.0	100.0	0	12054.6	58.3
f_{10}	43	0	14400.0	76.8	0	14400.0	94.8	0	14400.0	82.4	0	14400.0	100.0	**5**	10634.6	**44.2**
Total	416	7			3			4			3			**22**		

Table 7: Aggregate results for problem NLMCNDP-A

The results reported in Table 8 for the NLMCNDP-N show that our method shows a similar performance to `GUROBI`, with neither of them dominating the other. Although this problem seems to be less difficult than the MCNDNLA, it remains out of reach for about a 90% of the problems tested.

		Gurobi			**Couenne**			**SCIP**			**Naive**			**CN24**		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_1	43	1	13347.2	**27.7**	0	14400.0	91.4	0	14400.0	45.5	**6**	10162.9	52.0	4	9530.3	30.9
f_2	43	**7**	5836.2	**13.1**	0	14400.0	93.1	0	14400.0	74.9	6	10486.3	52.0	5	9022.7	32.7
f_3	42	3	10596.9	28.6	0	14400.0	88.1	2	11482.8	28.6	**4**	11204.3	64.1	3	11052.3	**27.6**
f_4	43	2	12075.7	31.5	0	14400.0	85.3	**4**	10742.9	**31.2**	2	12759.1	80.5	**4**	10821.1	43.0
f_5	36	**2**	9831.9	**41.9**	**2**	12979.3	77.2	**2**	10785.7	44.0	**2**	12605.2	77.2	**2**	10149.8	70.0
f_6	43	2	12677.9	**20.2**	0	14400.0	86.6	2	13419.9	35.4	**4**	11283.1	64.7	**4**	9360.6	30.8
f_7	43	**6**	8126.8	**11.2**	1	13942.4	81.7	4	10111.0	21.9	0	14400.0	100.0	4	10334.1	41.9
f_8	43	**14**	**3067.5**	**11.3**	2	12915.1	77.0	9	5715.8	17.1	11	6125.3	30.0	10	6380.9	22.1
f_9	43	**6**	7343.2	**9.7**	0	14400.0	87.2	2	12249.7	22.2	0	14400.0	100.0	5	9452.3	29.1
f_{10}	43	0	14400.0	**36.9**	0	14400.0	91.9	0	14400.0	51.5	0	14400.0	100.0	**3**	11088.7	48.3
Total	422	43			5			25			35			**44**		

Table 8: Aggregate results for problem NLMCNDP-N

5.7 Continuous knapsack

In the nonlinear continuous knapsack problem (NLCKP) we are given a collection of N objects and a knapsack of capacity K. Each object $u = 1 \dots N$ has a weight function $w_u(x)$ and a value function $g_u(x)$ for $x \in [0,1]$, representing the weight and reward associated to packing a fraction x of object u in the knapsack. The value $x = 1$ is therefore interpreted as packing the object entirely. To model this problem, we consider continuous variables $x_u \in [0,1]$ for every object u. The problem can be put

as the following NLP:

$$\texttt{maximize} \quad z = \sum_{u=1}^{N} g_u(x_u) \tag{34}$$

subject to

$$\sum_{u=1}^{N} w_u(x_u) \leq K \tag{35}$$

$$x \in [0,1]^N. \tag{36}$$

In this paper, we consider the special case where the functions w_u are linear, meaning that the function $w_u(x_u)$ can be written as $\omega_u x_u$ for a certain scalar ω_u.

Unlike the previous applications, the NLCKP involves maximizing a nonnegative reward function. Two modeling options arise when trying to accommodate the NLCKP to the proposed framework: 1) modify the objective to `minimize` $-z = -\sum_{u=1}^{N} g_u(x_u)$; or 2) consider upper bounding linear approximation functions to build the corridors, as opposed to lower bounding approximations. Both approaches are, however, equivalent. In our computational implementation, we have chosen the first modeling approach.

To assess the performance of our method, we have generated 70 problem instances following the recipe described in Trindade et al. (2023) for the same problem. We consider problems with $N \in \{10, 20, 50, 100, 200, 500, 1000\}$. For each N, we generate 10 random instances, hence the total of 70. Please note that in that paper, the authors also consider nonlinear cost functions similar in form to f_{10} but with problem-specific coefficients generated at random. Let us denote these cost functions $h_u, u = 1 \ldots N$. In our tests we omit the cost functions of the type f_{10} in favor of the functions of the type $\{h_u : u = 1 \ldots N\}$, that for nomenclature purposes we denote f_{11}. For all the cost functions except for f_{11}, the reward functions are defined as $g_u(x_u) = v_u f_i(x_u)$, where $v_u = h_u(1)$.

		Gurobi			**Couenne**			**SCIP**			**Naive**			**CN24**		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_1	70	18	3970.0	3.0	10	6423.6	10.6	19	3717.4	3.0	59	591.4	1.1	**60**	**464.4**	**0.5**
f_2	70	49	113.6	2.5	11	5807.5	21.4	13	5317.3	19.2	40	2952.7	6.2	**70**	**97.7**	**0.0**
f_3	70	22	1862.2	0.4	0	14400.0	22.4	20	2809.4	1.3	60	518.2	0.9	**70**	**22.3**	**0.0**
f_4	70	30	878.7	0.9	9	6520.8	30.4	20	2944.6	8.7	53	912.6	2.1	**70**	**22.7**	**0.0**
f_5	70	20	2269.8	0.5	0	14400.0	32.9	4	10985.4	10.8	46	1713.4	3.9	**70**	**39.4**	**0.0**
f_6	70	20	2393.2	26.0	6	8501.8	14.9	14	4001.6	3.4	50	1158.3	2.7	**57**	**855.3**	**0.9**
f_7	70	33	644.3	2.1	38	457.9	**0.0**	**70**	**3.6**	**0.0**	0	14400.0	100.0	**70**	13.3	**0.0**
f_8	70	**70**	**1.0**	**0.0**	69	39.8	0.1	**70**	6.2	**0.0**	49	756.7	3.0	**70**	14.0	**0.0**
f_9	70	26	1153.5	3.9	44	308.9	**0.0**	49	133.0	**0.0**	0	14400.0	100.0	**70**	**12.0**	**0.0**
f_{11}	68	11	5796.5	6.8	5	8906.8	25.9	18	2766.4	6.3	0	14400.0	100.0	**68**	**285.6**	**0.0**
Total	698	299			192			297			357			**675**		

Table 9: Aggregate results for problem NLCKP

From the results reported in Table 9 we observe a neat superiority of the proposed method, being able to solve more than twice as many problems as its closest competitors (`NAIVE`, `GUROBI` and `SCIP`, in that order), in computing times that are significantly lower (often by orders of magnitude) than for all the others.

6 Additional computational experiments

In this section we discuss two additional experiments. First, we test the scalability of our method for decreasing values of ϵ. Second, we analyze the performance of our method with respect to the types of the cost functions involved.

6.1 Scalability of the method to decreasing values of ϵ

We now assess the scalability of our method to decreasing values of ϵ (in %), starting from an initial tolerance of $\epsilon_0 = 10\%$. We analyze the performance of our method by measuring two indicators: the computing times required to converge within the desired tolerance and the number of iterations of the main loop to do so. The profiles curves depicted below are restricted to the problems successfully approximated to the desired tolerance, this is by omitting time outs.

In Figure 5 we plot profile graphs for two values of ϵ on the entirety of our testbed comprising several thousand problems. A point (x, y) in one of the profile curves is interpreted as the method being able to approximate Problem (1) within the desired tolerance for x problem instances in y seconds of computing time or less. Please note that the plot uses a semilogarithmic scale.

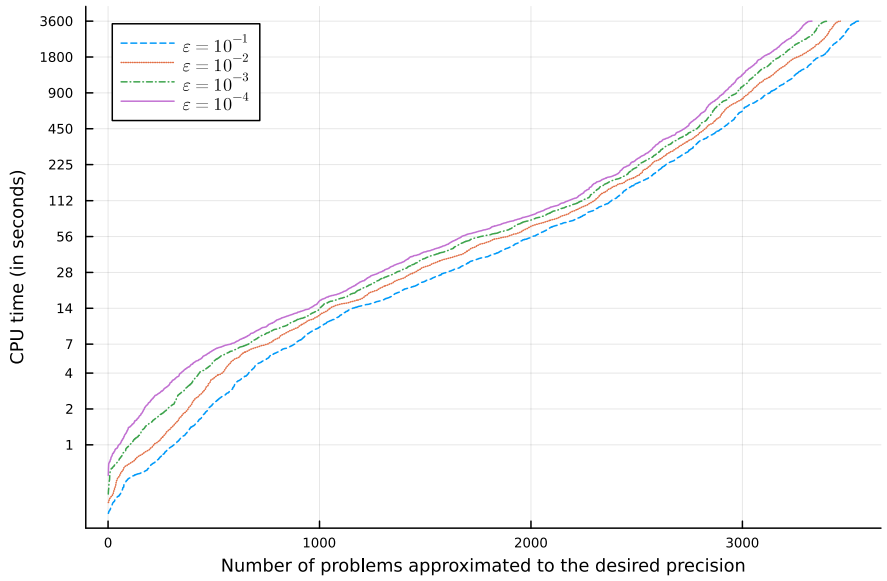


Figure 5: Profile graphs of the proposed methods for varying values of ϵ

This plot reveals an interesting feature of our method. It appears that transitioning from a rough tolerance to a much finer one (as for instance from $10^{-1}\%$ to $10^{-4}\%$) requires only a slightly higher computational effort. If this trend can be confirmed by future tests, it would mean that even much finer approximations could potentially be achieved without a much higher computational effort. Unfortunately, at this time our method cannot safely scale to finer tolerances as some of the subroutines (remarkably `LinA.jl` and `GUROBI`) start to fail and to show numerical instabilities due to floating point arithmetic errors.

Let us now look at the profile graph of the number of iterations of the main loop taken by our method to reach convergence to the desired tolerance, depicted in Figure 6. A point (x, y) in one of the profile curves is now interpreted as our method being able to approximate x problems to the desired tolerance within y iterations of the main loop of our method.

We observe an exponential growth of all the curves toward the end (i.e. for the harder problems), showing that an exponential trend is independent of the target tolerance. When restricted to the tightest tolerance of $\epsilon = 10^{-4}$ we also observe that the median occurs at 16 iterations (for about 1,800 problems approximated to the desired precision), with a few problems requiring more than 50 iterations and up to 124. If we compare both curves, we also observe that using a rougher precision of $\epsilon = 10^{-1}$ leads to a substantially lower number of iterations when compared to using the finer precision of $\epsilon = 10^{-4}$.

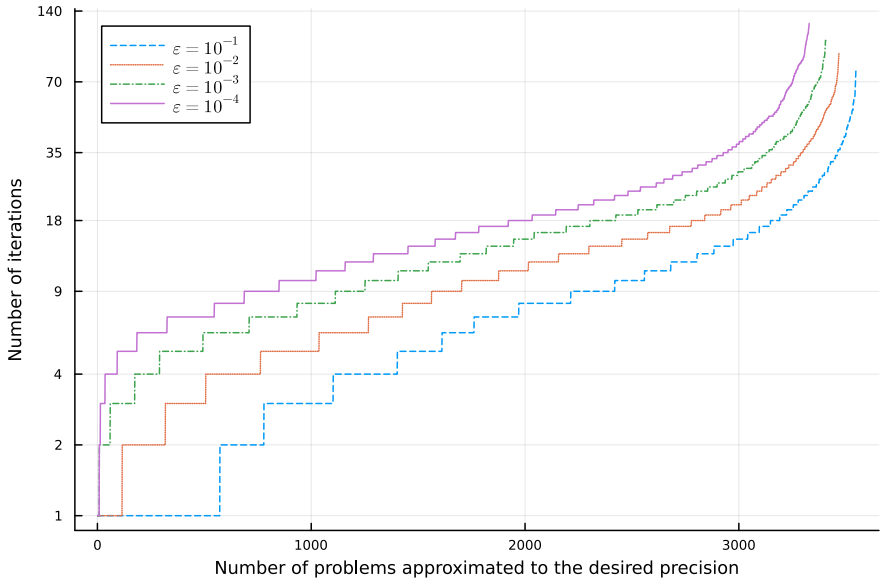


Figure 6: Number of iterations to reach convergence for varying values of ϵ

6.2 Performance on different cost functions

We now perform an analysis to understand how different attributes of the cost functions influence the performance of our method. We consider, aggregate in subgroups, cost functions that are either fixed-charge (f_1, f_2, f_6), pure concave (f_7, f_9), pure convex (f_8), monotonic with changes in concavity (f_3, f_4, f_6, f_{10}), or non-monotonic (f_5). Since these attributes may have different impacts depending on the sense of the optimization (as for instance in minimizing vs. maximizing a convex function), we restrict this analysis to minimization objectives, this is we omit from our analysis the NLCKP.

6.2.1 Fixed-charge functions

We consider a total of 1,564 problem instances for the three cost functions f_1, f_2, and f_6 with fixed-charge. The results in Table 10 show that our method solves the largest number of problems.

		Gurobi			**SCIP**			**Couenne**			**Naive**			**CN24**		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_1	526	321	**184.7**	2.6	321	488.5	3.2	171	2567.2	13.3	306	1235.3	5.9	**386**	421.2	**1.6**
f_2	510	161	2191.0	**3.0**	60	7850.5	45.0	105	5907.8	36.7	195	4397.5	16.3	**206**	**1816.8**	8.3
f_6	528	167	**1653.2**	15.6	31	10432.7	51.3	151	4501.3	25.3	94	7612.7	43.4	**217**	2379.2	**6.2**
Total	1564	649			412			427			595			**809**		

Table 10: Aggregate results for cost functions of the type fixed-charge

6.2.2 Pure concave functions

We again consider a total of 1,047 problem instances for the two cost functions f_7, f_9. The results in Table 11 show that our method solves more problems than `GUROBI`, although usually in about twice as much computing time.

		Gurobi			SCIP			Couenne			Naive			CN24		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_7	525	341	**248.5**	**1.9**	211	1162.4	8.4	203	2028.5	8.4	32	12527.0	75.2	**346**	533.7	2.1
f_9	522	307	**354.0**	2.4	192	1735.2	9.3	147	4669.9	15.0	4	14226.1	96.5	**344**	617.8	**2.0**
Total	1047	648			403			350			36			**690**		

Table 11: Aggregate results for cost functions of the type concave

6.2.3 Pure convex functions

We now consider a total of 506 problem instances for the cost function f_8. The results in Table 12 show that our method is overall dominated by GUROBI which is faster and solves significantly more problems. This also suggests that our method does not fully exploit convexities, and that further research should focus on better handling convexities.

		Gurobi			SCIP			Couenne			Naive			CN24		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_8	506	**433**	**38.6**	**0.6**	422	136.8	0.7	192	1370.1	11.7	286	836.9	6.5	251	758.2	2.3

Table 12: Aggregate results for cost functions of the type convex

6.2.4 Monotonic functions with changes in concavity

We now report our analysis for cost functions that are monotonic and that present changes in concavity, this is functions of the type f_3, f_4, f_6, and f_{10}. Our analysis considers a total of 2,110 problem instances. The results reported in Table 13 show that our solver is the one that performs best in terms of the number of problems solved, and of the final gaps.

		Gurobi			SCIP			Couenne			Naive			CN24		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_3	526	177	**1418.5**	10.9	173	1710.7	11.1	99	4734.3	21.6	**227**	2615.8	12.8	208	1552.0	**4.0**
f_4	528	180	**1401.2**	12.7	170	1865.2	14.8	85	5657.0	30.1	210	3445.3	15.1	**221**	2021.1	**6.0**
f_6	528	167	**1653.2**	15.6	31	10432.7	51.3	151	4501.3	25.3	94	7612.7	43.4	**217**	2379.2	**6.2**
f_{10}	528	158	1702.8	18.5	116	4219.3	25.4	158	2595.7	23.8	177	3391.1	20.5	**227**	**1282.2**	**2.7**
Total	2110	682			490			493			708			**873**		

Table 13: Aggregate results for cost functions of the type monotonic

6.2.5 Non-monotonic functions

We now consider a total of 517 problem instances for the cost function f_5. The results in Table 14 show that our method solves more problems than all the other methods, although GUROBI proves slightly faster.

		Gurobi			SCIP			Couenne			Naive			CN24		
Cost	#I	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP	#S	CPU	GAP
f_5	517	180	**1431.1**	**11.3**	168	2237.5	14.3	93	5266.6	29.3	203	4635.2	15.5	**209**	1922.4	11.7

Table 14: Aggregate results for cost functions of the type non-monotonic

7 Some limitations of our method

Our computational campaign has revealed some limitations of our method that we now discuss.

First, the number of separable nonlinear functions seems to play a key role in the scalability of our method. We observe that on problems with a very large number of separable nonlinear functions

like the NLUFLP-A, the NLTP, the NLMCFP-A, or the NLMCNDP-A —all of which including a quadratic number of nonlinear functions—, the performance of our method is often inferior than on those with a constant or linear number of nonlinear functions.

Second, we have noticed that symmetries play a nocive effect on the performance of our method. Because our method does not address symmetries, in the case where multiple optimal solutions exist in the domains of the nonlinear functions, our solver will oscillate by refining these zones in an alternate way. Hence, it may end up refining a significant portion of the entire domain before reaching the desired convergence.

Third, the results in Section 6.2 reveal that our method does not exploit convexities fully, being dominated by the other three solvers considered in our campaign when the functions f_i are convex (in a minimization setting).

8 Concluding remarks

We have introduced an iterative method with optimality guarantees for a general class of separable mixed-integer nonlinear problems. Our method iterates between the solution of a mixed-integer linear problem to compute primal and dual bounds, and a objective-driven repair procedure to tighten the bounds if deemed necessary. Our method does not rely on NLP nor MINLP oracles to compute those bounds —at the cost of being restricted to handle nonlinearities in the objective only—, hence relying exclusively on the efficiency to model and solve the resulting MILPs. We have proved that our method converges in a finite number of iterations under some very mild assumptions. We have assessed the effectiveness of our method on five optimization problems relevant in practice: the transportation problem, the uncapacitated facility location problem, the multicommodity flow problem, the multicommodity network design problem, and the continuous knapsack problem. Our results show that our method is efficient at handling these problems, often outperforming state-of-the-art solvers.

Future research shall focus on mitigating the nocive effects of symmetries as discussed in Section 7. Also, extending our framework to handle objectives that include functions of two or more variables that are not linearly separable in univariate functions would provide a significant contribution to the scientific literature. In addition, we believe that there is potential for applying some of the techniques introduced in this manuscript to nonlinear constraints, to make the method more generally applicable. The challenge is to maintain the guarantees of optimality, convergence and efficiency, while the polyhedral domain is modified at each iteration. Finally, we believe that the proposed method could benefit from specialized routines aiming at exploiting convexities.

Acknowledgments

We thank the Associate Editor and two anonymous reviewers for their constructive feedback that contributed in improving our presentation. C. Contardo thanks the Natural Sciences and Engineering Research Council (NSERC) of Canada for its financial support, under Grant no 2020-06311. S. U. Ngueveu thanks the FMJH Program PGMO for the financial support also provided by EDF-Thales-Orange.

References

A. A. Assad. Multicommodity network flows—a survey. *Networks*, 8(1):37–91, 1978.

R. E. Bank, A. H. Sherman, and A. Weiser. Some refinement algorithms and data structures for regular local mesh refinement. *Scientific Computing, Applications of Mathematics and Computing to the Physical Sciences*, 1:3–17, 1983.

J. E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4–5):597–634, 2009.

K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig. Enabling research through the SCIP Optimization Suite 8.0. *ACM Trans. Math. Softw.*, 49(2), jun 2023. ISSN 0098-3500. doi: 10.1145/3585516. URL `https://doi.org/10.1145/3585516`.

J. Bey. Simplicial grid refinement: On freudenthal's algorithm and the optimal number of congruence classes. *Numerische Mathematik*, 85(1):1–29, 2000.

N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. The continuous-time service network design problem. *Operations Research*, 65(5):1303–1321, 2017. doi: 10.1287/opre.2017.1624. URL `https://doi.org/10.1287/opre.2017.1624`.

N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh. The price of discretizing time: a study in service network design. *EURO Journal on Transportation and Logistics*, 8(2):195–216, 2019. doi: 10.1007/s13676-018-0119-x. URL `https://doi.org/10.1007/s13676-018-0119-x`.

R. Burlacu. On refinement strategies for solving MINLPs by piecewise linear relaxations: a generalized red refinement. *Optimization Letters*, 2021. doi: 10.1007/s11590-021-01740-1. URL `https://doi.org/10.1007/s11590-021-01740-1`.

R. Burlacu, B. Geißler, and L. Schewe. Solving mixed-integer nonlinear programmes using adaptively refined mixed-integer linear programmes. *Optimization Methods and Software*, 35(1):37–64, 2020. doi: 10.1080/10556788.2018.1556661. URL `https://doi.org/10.1080/10556788.2018.1556661`.

J. Codsi, B. Gendron, and S. U. Ngueveu. Lina: A faster approach to piecewise linear approximations using corridors and it's application to mixed integer optimization. Technical report, LAAS-ROC, 2021. URL `https://hal.archives-ouvertes.fr/hal-03336003`.

G. Cornuéjols and J.-M. Thizy. Some facets of the simple plant location polytope. *Mathematical programming*, 23(1):50–74, 1982.

T. G. Crainic, A. Frangioni, and B. Gendron. Bundle-based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics*, 112(1-3):73–99, 2001.

C. D'Ambrosio, J. Lee, and A. Wächter. A global-optimization algorithm for mixed-integer nonlinear programs having separable non-convexity. In *European Symposium on Algorithms*, pages 107–118. Springer, 2009.

C. D'Ambrosio, J. Lee, and A. Wächter. An algorithmic framework for MINLP with separable non-convexity. In *Mixed Integer Nonlinear Programming*, pages 315–347. Springer, 2012.

C. D'Ambrosio, A. Frangioni, and C. Gentile. Strengthening the sequential convex MINLP technique by perspective reformulations. *Optimization Letters*, 13(4):673–684, 2019.

J. E. Ertel and E. B. Fowlkes. Some algorithms for linear spline and piecewise multiple linear regression. *Journal of the American Statistical Association*, 71(355):640–648, 1976.

L. R. Ford Jr and D. R. Fulkerson. Solving the transportation problem. *Management Science*, 3(1): 24–32, 1956.

H. Freudenthal. Simplizialzerlegungen von beschrankter flachheit. *Annals of Mathematics*, 43(3): 580–582, 1942. ISSN 0003486X. URL http://www.jstor.org/stable/1968813.

B. Geißler, A. Martin, A. Morsi, and L. Schewe. Using piecewise linear functions for solving MINLPs. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, pages 287–314, New York, NY, 2012. Springer New York. ISBN 978-1-4614-1927-3.

O. Günlük and J. Linderoth. Perspective reformulations of mixed integer nonlinear programs with indicator variables. *Mathematical programming*, 124:183–205, 2010.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL https://www.gurobi.com.

J. Harkness and C. ReVelle. Facility location with increasing production costs. *European Journal of Operational Research*, 145(1):1–13, 2003. ISSN 0377-2217. doi: https://doi.org/10.1016/S0377-2217(02)00176-5. URL https://www.sciencedirect.com/science/article/pii/S0377221702001765.

K. Holmberg. Exact solution methods for uncapacitated location problems with convex transportation costs. *European Journal of Operational Research*, 114(1):127–140, 1999. ISSN 0377-2217.

K. Holmberg, M. Rönnqvist, and D. Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113(3):544–559, 1999.

J. Huchette and J. P. Vielma. Nonconvex piecewise linear functions: Advanced formulations and simple modeling tools, 2019.

F. Hwang and Y. Huang. An effective logarithmic formulation for piecewise linearization requiring no inequality constraint. *Computational Optimization and Applications*, 79:601 – 631, 2021.

IBM. *CPLEX Optimization Studio 20.1*, 2020.

P. Kleinschmidt and H. Schannath. A strongly polynomial algorithm for the transportation problem. *Mathematical Programming*, 68(1-3):1–13, 1995.

D. Lu, F. Gzara, and S. Elhedhli. Facility location with economies and diseconomies of scale: models and column generation heuristics. *IIE Transactions*, 46(6):585–600, 2014. doi: 10.1080/0740817X.2013.860508. URL https://doi.org/10.1080/0740817X.2013.860508.

L. Marshall, N. Boland, M. Savelsbergh, and M. Hewitt. Interval-based dynamic discretization discovery for solving the continuous-time service network design problem. *Transportation science*, 55 (1):29–51, 2021.

R. Martinelli, D. Pecin, and M. Poggi. Efficient elementary and restricted non-elementary route pricing. *European Journal of Operational Research*, 239(1):102–111, 2014.

H. Nagarajan, M. Lu, E. Yamangil, and R. Bent. Tightening McCormick relaxations for nonlinear programs via dynamic multivariate partitioning. In *International Conference on Principles and Practice of Constraint Programming*, pages 369–387. Springer, 2016. doi: 10.1007/978-3-319-44953-1_24.

H. Nagarajan, M. Lu, S. Wang, R. Bent, and K. Sundar. An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs. *Journal of Global Optimization*, 2019. ISSN 1573-2916. doi: 10.1007/s10898-018-00734-1.

S. U. Ngueveu. Piecewise linear bounding of univariate nonlinear functions and resulting mixed integer linear programming-based solution methods. *Euopean Journal of Operational Research*, 275:1058–1071, 2019.

J. O'Rourke. An on-line algorithm for fitting straight lines between data ranges. *Communications of the ACM*, 24(9):574–578, 1981.

D. C. Paraskevopoulos, S. Gürel, and T. Bektaş. The congested multicommodity network design problem. *Transportation Research Part E: Logistics and Transportation Review*, 85:166–187, 2016.

S. Rebennack and J. Kallrath. Continuous piecewise linear delta-approximations for univariate functions: Computing minimal breakpoint systems. *Journal of Optimization Theory and Applications*, 167(2):617–643, 2015.

S. Rebennack and V. Krasko. Piecewise linear function fitting via mixed-integer linear programming. *INFORMS Journal on Computing*, to appear, 2019.

G. Righini and M. Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal*, 51(3):155–170, 2008.

B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020. doi: 10.1007/s12532-020-00179-2. URL `https://doi.org/10.1007/s12532-020-00179-2`.

I. Tomek. Piecewise-linear approximation with a bound on absolute error. *Computers and Biomedical Research*, 7(1):64–70, 1974a.

I. Tomek. Two algorithms for piecewise-linear continuous approximation of functions of one variable. *IEEE Transactions on Computers*, 100(4):445–448, 1974b.

R. S. Trindade, C. d'Ambrosio, A. Frangioni, and C. Gentile. Comparing perspective reformulations for piecewise-convex optimization. *Operations Research Letters*, 51(6):702–708, 2023.

J. P. Vielma and G. L. Nemhauser. Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. *Mathematical Programming*, 128(1):49–72, 2011.

J. P. Vielma, S. Ahmed, and G. Nemhauser. Mixed-integer models for nonseparable piecewise-linear optimization: unifying framework and extensions. *Operations Research*, 58(2):303–315, 2010.

A Relative performance against `Alpine.jl`

In this appendix we perform a comparative analysis of the different methods considered in our main computational campaign against the iterative method of Nagarajan et al. (2019) and its open-source implementation for the Julia language named `Alpine.jl`. We detail these results because Nagarajan et al. also apply an adaptive partitioning and decompose the domain into non-uniform subdomains. This comparison illustrates to which extent the choices made for our method are appropriate for solving problem (1).

Note that, given that both methods are natively implemented in Julia, the comparisons are not influenced by external attributes such as the choice of the computing architecture, programming language, operating system, etc. Instead, it is safe to assume that any significant difference between the methods reflects their relative algorithmic performance for the problems tested. Also, please note that currently `Alpine.jl` can only handle polynomial expressions in the objectives or constraints, and as such we restrict or analysis to the objectives f_2, f_8 as described in Section 5.1. The tests have

been executed on the same machine and architecture as for our previous tests, with a time limit of one hour per run. The differences in performance as you will see below are so remarkable that we restrict this analysis to the NLCKP.

		Baron		**SCIP**		**Gurobi**		**Naive**		**CN24**		**Alpine**	
N	#I	#S	CPU	#S	CPU	#S	CPU	#S	CPU	#S	CPU	#S	CPU
10	10	6	203.2	**10**	8.7	**10**	**0.2**	**10**	198.0	**10**	8.8	4	2995.7
20	10	7	132.1	3	8802.0	**10**	**0.2**	**10**	421.1	**10**	17.5	5	2745.8
50	10	4	1789.2	0	14400.0	**10**	**0.5**	**10**	1059.6	**10**	41.7	3	7437.0
100	10	0	14400.0	0	14400.0	**10**	**6.3**	**10**	2109.7	**10**	84.6	0	14400.0
200	10	0	14400.0	0	14400.0	9	**110.7**	0	14400.0	**10**	164.9	0	14400.0
500	10	0	14400.0	0	14400.0	0	14400.0	0	14400.0	**10**	**425.2**	0	14400.0
1000	10	0	14400.0	0	14400.0	0	14400.0	0	14400.0	**10**	**860.7**	0	14400.0
Total	70	17		13		49		40		**70**		12	

Table 15: Comparison against `Alpine.jl` on problem NLCKP and cost functions f_2

		Baron		**SCIP**		**Gurobi**		**Naive**		**CN24**		**Alpine**	
N	#I	#S	CPU	#S	CPU	#S	CPU	#S	CPU	#S	CPU	#S	CPU
10	10	**10**	2.8	**10**	**0.1**	**10**	0.1	**10**	89.3	**10**	1.1	**10**	105.5
20	10	**10**	2.8	**10**	0.2	**10**	**0.1**	**10**	92.0	**10**	1.7	**10**	46.6
50	10	**10**	3.0	**10**	0.4	**10**	**0.2**	**10**	376.1	**10**	4.1	7	585.9
100	10	**10**	3.1	**10**	0.8	**10**	**0.2**	9	1995.9	**10**	7.3	8	360.1
200	10	**10**	4.1	**10**	1.3	**10**	**0.4**	0	14400.0	**10**	12.8	6	1033.1
500	10	**10**	12.4	**10**	4.8	**10**	**1.0**	0	14400.0	**10**	35.2	5	3972.3
1000	10	**10**	74.5	**10**	21.3	**10**	**5.8**	0	14400.0	**10**	129.9	5	3753.1
Total	70	**70**		**70**		**70**		39		**70**		51	

Table 16: Comparison against `Alpine.jl` on problem NLCKP and cost functions f_8

As we can observe, `Alpine.jl`'s performance is often the worst, being capable of solving only a few problems within the desired precision, and in very high computing times.

B Detailed computational results

In this section we present detailed computational results. In Tables 17-136 we report, for every problem instance considered in our campaign, the dual bound (under column `DB`), primal bound (under column `PB`), relative gap (under column `GAP`), and CPU time in seconds (under column `CPU`). For the solvers `CN24` and `NAIVE` in addition we report the total number of linear pieces (under column `NP`). For our solver, in addition we report the number of iterations of the main loop of our method (under column `NIT`). We also report the total number of problems solved and the shifted geometric means of the computing times, and the gaps, in each table.