Spatial Branch-and-Bound for Nonconvex Separable Piecewise Linear Optimization

Thomas Hübner, Akshay Gupte, Steffen Rebennack

Please scroll down for article—it is on subsequent pages

Spatial Branch-and-Bound for Nonconvex Separable Piecewise Linear Optimization

Thomas Hübner,[a] Akshay Gupte,[b,c] Steffen Rebennack[d,*]

[a] Power Systems Laboratory, ETH Zürich, 8092 Zurich, Switzerland; [b] School of Mathematics, University of Edinburgh, Edinburgh EH9 3FD, United Kingdom; [c] Maxwell Institute for Mathematical Sciences, University of Edinburgh, Edinburgh EH8 9BT, United Kingdom; [d] Institute for Operations Research, Stochastic Optimization, Karlsruhe Institute of Technology, 76185 Karlsruhe, Germany
*Corresponding author
Contact: thuebner@ethz.ch (TH); akshay.gupte@ed.ac.uk (AG); https://orcid.org/0000-0002-7839-165X (AG); steffen.rebennack@kit.edu, https://orcid.org/0000-0002-8501-2785 (SR)

Abstract. Nonconvex separable piecewise linear functions (PLFs) frequently appear in applications and to approximate nonlinearitites. The standard practice to formulate nonconvex PLFs is from the perspective of discrete optimization using special ordered sets and mixed-integer linear programs (MILPs). In contrast, we take the viewpoint of global continuous optimization and present a spatial branch-and-bound algorithm for optimizing a separable discontinuous PLF over a closed convex set. It offers slim and sparse linear programming relaxations, sharpness throughout the search tree, and an increased flexibility in branching decisions. The main feature of our algorithm is the generation of convex underestimators at the root node of the search tree and their quick and efficient updates at each node after branching. Convergence to the global optimum is achieved when the PLFs are lower semicontinuous. A Python implementation of our algorithm is tested on knapsack and network flow problems for both continuous and discontinuous PLFs. Our algorithm is compared with four logarithmic MILP formulations solved by Gurobi's MILP solver as well as Gurobi's PLF solver. We also compare our method against mixed-integer nonlinear program formulations solved by Gurobi. The numerical experiments indicate significant performance gains up to two orders of magnitude for medium- to large-sized PLFs. Finally, we also give an upper bound on the additive error from PLF approximations of nonconvex separable optimization.

1. Introduction
1.1. Literature Review

A piecewise linear function (PLF) is a multivariate function whose domain can be partitioned into pieces such that the function is affine in each piece. Such a nonsmooth function arises naturally in some optimization problems or more commonly, as an approximation of a nonlinear, nonconvex function (Geisler et al. 2012, Dey and Gupte 2015, Nagarajan et al. 2019, Burlacu et al. 2020, Beach et al. 2022, Bärmann et al. 2023, Warwicker and Rebennack 2024). When a PLF is convex and is either minimized or appears in a $\leqslant$ constraint, it can be modeled as a linear program (LP). In general, a PLF is NP-hard to optimize (Keha et al. 2006) even for separable PLFs, which can be written as a sum of univariate PLFs each of which is in a different coordinate. Separable PLFs appear naturally in a wide variety of problems in various fields dealing with economies of scale, such as logistics, management, finance, or engineering (Markowitz and Manne 1957, Dantzig 1960, Beale and Forrest 1976). Univariate PLFs also

arise as approximations to one-dimensional nonconvex functions in a global optimization problem (Leyffer et al. 2008, Natali and Pinto 2009, Rebennack and Kallrath 2015, Grimstad and Knudsen 2020, Posypkin et al. 2020, Sundar et al. 2022). In fact, a separable concave function minimization can be approximated to an arbitrary precision by a single separable PLF problem (Magnanti and Stratila 2004).

The common way to approach problems with nonconvex PLFs is by developing exact formulations based on either *mixed-integer linear programs* (MILPs) or *special ordered sets of type 2* (SOS2). In both approaches, the problem is reformulated by using a number of additional variables, some of which are binary, and constraints for each breakpoint of the PLF. This reformulation is then solved with an MILP solver. Classical MILP and SOS2 modeling approaches (see surveys in Vielma et al. 2010 and Rebennack 2016) initially focused on continuous separable PLFs but were later extended by Vielma et al. (2010) to the nonseparable case. It is known that their LPs provide the same relaxation strength (Sherali 2001, Croxton et al. 2003, Keha et al. 2004). However, they have the drawback of using as many binary variables as the number of segments of a PLF. This was remedied by Vielma and Nemhauser (2011) and Huchette and Vielma (2023), who produced MILPs that require only a logarithmic number of binary variables, thereby allowing for greater scalability of such models. Other research has focused on specialized valid inequalities for the SOS2-based models of separable PLFs (Keha et al. 2006, Vielma et al. 2008, de Farias et al. 2013, Zhao and de Farias 2013). Extensions of MILP models to lower semicontinuous (l.s.c.) PLFs have been studied (Vielma et al. 2008, 2010). For general discontinuous PLFs, one cannot expect an MILP formulation with bounded integer variables (Meyer 1976, theorem 2.1), but the SOS2 branching scheme has been adapted (de Farias et al. 2008). Many of these modeling and algorithmic advances have been implemented in state-of-the-art MILP solvers and leveraged to build stronger polyhedral relaxations of nonconvex functions (Rebennack 2016, Kim et al. 2024, Lyu et al. 2025).

Because a PLF is a nonconvex function, the problem of optimizing a PLF can be viewed through the lens of global optimization. A commonly used algorithmic framework for global optimization is spatial branch-and-bound (sBB). The use of an sBB for optimizing a separable function (sum of univariate functions, not necessarily PLF) was first done by Falk and Soland (1969). This was improved upon by Horst (1986) and Tuy and Horst (1988) to general nonconvex functions; since then, sBB algorithms for global optimization have matured immensely (cf. Locatelli and Schoen 2013, Tuy 2016), and there are many sophisticated implementations in global solvers for optimizing smooth functions. However, this global optimization approach has so far not been undertaken for PLF optimization, and state-of-the-art global solvers are unable to take PLFs directly as input without first being modeled using integer variables as mentioned above. Another drawback of existing methods is that they do not always scale well with the number of segments in the PLF. We adopt the global optimization approach, and our experiments show that the sBB approach has better scalability properties than the MILP or SOS2 models.

1.2. Our Contributions

We study the global optimization of a separable nonconvex PLF over a closed convex set. Contrary to the standard combinatorial approach of using an MILP or special ordered sets formulation to model the PLF, we take the *nonlinear* approach to solving such problems. We do not reformulate the PLF with integer variables, but instead, we generate convex underestimators for it and refine them to develop an sBB algorithm. A key ingredient of our algorithm is how the underestimator is generated even when the PLF is discontinuous and how it is efficiently and quickly updated at a child node using the information from the parent node and without having to generate it from scratch. Our contribution of adding a new method to the literature complements the MILP and SOS2 approaches by offering the following advantages: (i) slim and sparse LP relaxations, (ii) sharpness throughout the search tree, and (iii) more freedom in branching decisions. Through extensive computational experiments, we demonstrate that even a rudimentary Python implementation of the sBB can provide speedups of two orders of magnitude over modern logarithmic models solved by Gurobi if the number of segments is sufficiently large and that these speedups tend to grow with every segment added to the PLFs.

The existing approaches for PLF optimization use integer branch-and-bound (B&B), where branching takes place on integer (mostly binary) variables in a binary search tree and bounding is through LP relaxations (enhanced with cutting planes). Our sBB also uses LP relaxations (albeit of a different kind) for bounding but branches on continuous variables only (hence, the term *spatial*). Hence, finite convergence to the global optimum is not obvious with our approach and in fact, is not possible for all branching rules. We provide a rule that branches only at the breakpoints and enables the sBB to converge finitely. The classical largest-error branching rule is known to converge asymptotically for a continuous separable objective, and we present an independent and self-contained proof using Lipschitz continuity of PLFs. For general objective functions that are either lower semicontinuous or such that their values at infeasible points are no lower than the global minimum, the longest-edge branching rule has been shown to achieve asymptotic convergence, and this also carries over to our PLF

optimization problem. The lack of finite convergence for any branching rule could be perceived as a drawback of sBB versus integer B&B, which always terminates finitely for bounded integer variables. However, our experiments show that this convergence issue arises only when the number of segments in a PLF is small, and the sBB generally terminates quicker for larger instances.

To the best of our knowledge, the various sBB-based state-of-the-art global solvers cannot handle PLFs directly unless they are explicitly input to the solver formulated as MILPs. Thus, we see our work as a first step in the direction of creating an sBB solver that can optimize a separable PLF without creating integer variables. We begin in Section 2 by describing the problem input and basics of sBB from the literature. Section 3 introduces the basic concepts of our sBB algorithm and relates it to the MILP and SOS2 approaches. Section 4 studies various convexification properties of a univariate PLF that underpin our algorithm. The sBB algorithm, with all of its elements, is described and analyzed for convergence in Section 5. Computational testing is done in Section 6, where comparisons are also drawn with logarithmic-sized MILP models, Gurobi's PLF solver, and Gurobi's global solver. Section 7 derives a bound on the number of segments necessary in a good PLF approximation of separable Hölder-continuous functions. Lastly, conclusions and some future directions are mentioned in Section 8.

1.3. Importance of Scalable Algorithms

Because our experiments show the sBB to have better computational performance than MILP or SOS2 models as the number of segments in the PLF increases, we briefly discuss here the importance of a method with such good scalability properties.

PLFs are commonly employed to linearize nonlinear terms and thereby, create a tractable approximation to a nonconvex optimization problem. PLF approximations can be constructed either as relaxations (outer approximations) or through discretizations (inner approximations). When the nonconvexities are present in the constraints, the PLF approximation resulting from discretization may not necessarily produce an inner approximation of the feasible region but can nonetheless be used to obtain some approximate solution. Small segments in the PLF give fine approximations of the problem, which may translate into sharp primal or dual bounds. Thus, a key question when building PLF approximations is to determine how many pieces each PLF should have if the approximation error, defined as the largest distance between the function value and the approximate value, is to be no more than some given error bound. We mention some results for a continuous univariate function over a closed interval because that is the focus of this paper, but we note that some error-bounding analysis has also been done for higher-dimensional functions (Dey and Gupte 2015, Adams et al. 2019, Duguet and Ngueveu 2022, Bärmann et al. 2023).

The errors in the PLF relaxation of a univariate function are an elementary calculation because this relaxation is constructed by first partitioning the interval into alternate regions of convexity and concavity for the function and then, drawing tangents at different points in the convex regions and drawing secants in the concave regions. The analysis is nontrivial for the case of the PLF approximation, which is constructed by choosing some breakpoints in the interval (either equidistant or not) and connecting them at their function values. For this discretization, Frenzen et al. (2010, theorems 1 and 2) gave an asymptotic answer by showing that for thrice-continuously differentiable functions, the number of breakpoints to achieve an error of ε is roughly $c/\sqrt{\varepsilon}$ as $\varepsilon \to 0$, where the constant c depends on the second-order derivative of the function. Another related question is to consider optimization of a separable function and determine the number of breakpoints necessary to construct a PLF approximation whose optimal value is no worse than a given tolerance ε away from the true optimum. Such bounds on the number of segments have been derived using first- and second-order derivatives when the function is convex or concave (Thakur 1978, Kontogiorgis 2000, Magnanti and Stratila 2004). When the objective function is nonseparable, it is possible to construct separable PLF underestimators and use their error bounds to obtain a globally convergent algorithm (Feijoo and Meyer 1988).

There are computationally intensive MILP-based methods for computing best-fit PLFs (Toriello and Vielma 2012, Ngueveu 2019, Kong and Maravelias 2020, Rebennack and Krasko 2020, Warwicker and Rebennack 2022) as well as efficient algorithms (Warwicker and Rebennack 2024). Even if logarithmically many binary variables are used, the number of continuous variables generally scales linearly with the number of pieces in each function. Therefore, in order to obtain tight approximations of nonlinear functions, large-sized MILPs have to be solved, and branch-cut algorithms do not always converge very quickly on these. Recognizing this obstacle, some recent studies (Nagarajan et al. 2019, Burlacu et al. 2020, Gupte et al. 2022) have looked at algorithms that adapt the location of the breakpoints in the PLF approximation so that large-sized mixed-integer formulations do not have to be created a priori; but, their results are far from conclusive, and there is still scope for devising new methods with better scalability.

2. Preliminaries

2.1. Problem Input

We consider the separable nonconvex piecewise linear optimization problem given by

$$\mathcal{P}: \quad v^* = \inf F(x) := \sum_{i=1}^{n} f_i(x_i), \quad \text{s.t.} \ \ x \in S \cap H, \tag{1}$$

where every $f_i : [l_i, u_i] \to \mathbb{R}$ is a univariate PLF, possibly nonconvex and discontinuous, over an interval $H_i := [l_i, u_i]$. Some of the f_i's can be constant functions. The feasible set is the intersection of a closed convex set $S \subset \mathbb{R}^n$ and a hyperrectangle $H := \{x \in \mathbb{R}^n : l_i \leqslant x_i \leqslant u_i, i = 1, \ldots, n\}$. When each f_i is l.s.c. over $[l_i, u_i]$, the problem is solvable in the sense that the optimal value v^* is attained by some feasible solution. For general discontinuous functions, optimal solutions may not exist, and so, we can only hope to find v^*. Note that when H is not given explicitly in the description of the feasible set, variable bounds can be computed if S is compact. For simplicity and ease of notation, we assume that the intervals in each coordinate satisfy $H_i =$ projection of $S \cap H$ onto x_i. This can be achieved after some preprocessing and optimality-based bound tightening techniques.

Each PLF f_i is input with its $K_i + 1$ breakpoints in $[l_i, u_i]$ for some integer $K_i \geqslant 1$, and these are indexed by the set $\mathcal{K}_i := \{0, 1, \ldots, K_i\}$. The breakpoints include the two endpoints l_i and u_i and the points where f_i either changes its slope or is discontinuous. Denote the x values of the breakpoints by

$$B_i := \{b_i^k : k \in \mathcal{K}_i\}, \quad \text{with} \quad l_i = b_i^0 < b_i^1 < b_i^2 < \cdots < b_i^{K_i} = u_i. \tag{2a}$$

The function values at the breakpoints are $\{y_i^k : k \in \mathcal{K}_i\}$. Because we allow discontinuities at the breakpoints, we also need to know the left and right limits at each breakpoint to characterize f_i. The left limit is denoted by $y_i^{k,-}$, and the right limit is denoted by $y_i^{k,+}$. For the left (respectively, right) endpoint, we set the left (respectively, right) limit to the function value. Thus, for every $k \in \mathcal{K}_i$, we have as input the tuple

$$(b_i^k, y_i^k, y_i^{k,-}, y_i^{k,+}).$$

Using this input, a univariate PLF can be defined over $[b_i^k, b_i^{k+1})$, for any $k \in \mathcal{K}_i$, as

$$f_i(x_i) = \begin{cases} y_i^k, & x_i = b_i^k \\ \dfrac{y_i^{k+1,-} - y_i^{k,+}}{b_i^{k+1} - b_i^k}(x_i - b_i^k) + y_i^{k,+}, & b_i^k < x_i < b_i^{k+1}. \end{cases} \tag{2b}$$

If f_i is continuous at a breakpoint b_i^k (i.e., $y_i^k = y_i^{k,-} = y_i^{k,+}$), we write (b_i^k, y_i^k), knowing that the left and right limits coincide with the function value.

2.2. Background on sBB

The spatial branch-and-bound is similar to the integer branch-and-bound, but it has some major differences. In sBB, lower bounds are computed by a convex relaxation (convexification), which is obtained after replacing every nonconvex function by a *convex underestimator* over its bounded function domain. The strength of relaxations is important for convergence of the algorithm, and a fast numerical performance depends on the speed and efficiency with which the relaxations are generated and updated throughout the branching tree. Second, branching takes place on continuous variables, which leads to a partition of the feasible region in hyperrectangles. Third, after branching has occurred and any bound tightening has been performed on the variables, the underestimator is updated and refined to obtain a stronger relaxation than what is implied by the original relaxation with new variable bounds on it. Convergence in limit to the global optimum can then be obtained under mild conditions and assumption of lower semicontinuity of the functions because branching results into smaller hyperrectangles, which allow for tighter underestimators that force the gap between the function and its underestimator to converge to zero. The reader is referred to Locatelli and Schoen (2013, chapter 5.4) for a more detailed description of the general convergence theory of sBB algorithms. It is known that for optimizing any nonconvex function over a closed convex set, an sBB algorithm converges in finitely many iterations for any $\varepsilon > 0$ optimality tolerance if the following two properties are satisfied:

1. *exhaustiveness* of branching (which means that any nested infinite subsequence of hyperrectangles used for branching converges to a point) and

2. *exactness in the limit* for the underestimators (which means that their gap to the function value at any point goes to zero as the branching hyperrectangles shrink to a point).

With $\varepsilon = 0$, only convergence in the limit is guaranteed if besides the above two properties, the sBB also selects nodes infinitely often using the best bound rule. Some of the branching rules can also lead to finite convergence with $\varepsilon = 0$ if there is some special structure on the optimal solutions, such as an extreme point property (Shectman and Sahinidis 1998, Al-Khayyal and Sherali 2000).

3. Overview of Our sBB

3.1. Main Ideas

There are two main components to our sBB—convex relaxations using underestimators to obtain lower bounds and branching rules to guarantee convergence to global optimum. We do not employ any heuristics, and so, upper bounds are calculated in the standard way of evaluating the value of F at a solution to a node relaxation in the sBB search tree. One could possibly obtain stronger upper bounds by employing derivative-free optimization algorithms to minimize F using the node relaxation solution as a starting point, but exploring this idea is out of scope for this paper. Our branching rules are adopted from the literature and explained later in Section 5.2. In the remainder of this section, we outline our convex relaxation.

The convex envelope of a function over a compact convex set is defined as the point-wise supremum of all of the convex underestimators of that function over the set. Minimizing the function over the set is equivalent to minimizing its convex envelope. However, this envelope is generally intractable to compute, and the same is also true for nonconvex PLFs. The difficulty generally arises from the presence of the set S, which could be nontrivial and complicated, and so, the standard approach in global optimization is to generate convex underestimators of the objective function over the hyperrectangle H instead of over $S \cap H$. Because H is the Cartesian product of one-dimensional convex compact intervals and F is a separable function, the envelope of F over H is a sum of univariate envelopes. Using cvx to denote the convex envelope operator, we can write $\text{cvx}_H F(x) = \sum_{i=1}^{n} \text{cvx}_{H_i} f_i(x_i)$. Each $\text{cvx}_{H_i} f_i$ is a PLF, but because f_i is allowed to be discontinuous, this PLF may not be l.s.c. For computational tractability, we need the underestimators to be l.s.c. so that they have a polyhedral representation; otherwise, the corresponding feasible set of the relaxation will not be a closed set, which creates numerical difficulties in solving this relaxation. Hence, we carry out one additional step for the underestimators. For each i, we take the envelope of an l.s.c. function underestimating f_i. The resulting function is not only convex and l.s.c. but in fact, convex and continuous because of convex functions being u.s.c. over polytopes. Let us denote this underestimator for each i by $\text{vex}_{H_i} f_i$. Summing these yields a convex continuous PLF underestimator on F,

$$\text{vex}_H F(x) := \sum_{i=1}^{n} \text{vex}_{H_i} f_i(x_i), \quad x \in H. \tag{3a}$$

This yields a convex relaxation for Problem (1) whose value we denote by $\underline{v}(H)$:

$$v^* \geqslant \underline{v}(H) := \inf_x \sum_{i=1}^{n} \text{vex}_{H_i} f_i(x_i) \quad \text{s.t.} \ x \in S \cap H \tag{3b}$$

$$= \inf_{x,z} \sum_{i=1}^{n} z_i \quad \text{s.t.} \ \text{vex}_{H_i} f_i(x_i) \leqslant z_i, \ x \in S \cap H, \tag{3c}$$

where the second equality is from using the epigraph modeling step.

Because each $\text{vex}_{H_i} f_i$ is a convex continuous PLF, its epigraph is a polyhedron, and so, $\text{vex}_{H_i} f_i$ is equal to the point-wise maximum of finitely many affine functions. Thus, there is a finite set $\mathcal{E}_i(H)$ and coefficients (a_{ik}, b_{ik}) for $k \in \mathcal{E}_i(H)$ such that

$$\text{vex}_{H_i} f_i(x_i) = \max_{k \in \mathcal{E}_i(H)} a_{ik} x_i + b_{ik}, \qquad x_i \in H_i.$$

Our construction of $\text{vex}_{H_i} f_i$ is such that $\mathcal{E}_i(H) \subseteq \mathcal{K}_i$ with $\{0, K_i\} \subseteq \mathcal{E}_i(H)$, where we recall from (2a) that $\mathcal{K}_i$ indexes the breakpoints of f_i. Hence, the coefficients (a_{ik}, b_{ik}) for each $k \in \mathcal{E}_i(H)$ can be obtained in terms of the values of f_i at these breakpoints. Therefore, our convex relaxation of problem $\mathcal{P}$ is as follows:

$$v^* \geqslant \underline{v}(H) = \min \ \sum_{i=1}^{n} z_i \tag{4a}$$

$$\text{s.t.} \ a_{ik} x_i + b_{ik} \leqslant z_i, \qquad k \in \mathcal{E}_i(H), \ i = 1, \ldots, n \tag{4b}$$

$$x \in S \cap H. \tag{4c}$$

A salient feature of this work is the efficient computation of the underestimator $\text{vex}_{H_i} f_i$, and this is presented in Algorithm 4.1. However, this only represents the root-node relaxation. In the search tree of sBB, H is successively partitioned into a sequence of hyperrectangles $H^t \subset H$, and so, the relaxation (4) has to be constantly updated and solved again over $S \cap H^t$. In order to make our algorithm competitive and efficient, for any $H^t \subset H$, we do not compute the lower bound $v(H^t)$ by computing $\text{vex}_{H_i^t} f_i$ from scratch using the breakpoints of f_i, although this is certainly an option. Instead, we update the underestimator that was computed for the parent node of the node corresponding to H^t by exploiting structural properties of PLFs. Let us elaborate on this point. If H^s is the hyperrectangle for the parent node of the node for H^t and x_{i_t} was the branching variable used to create H^t from H^s, then our underestimators at the two nodes differ only in the coordinate x_{i_t} so that

$$\text{vex}_{H^t} F(x) = \left[\sum_{i \neq i_t} \text{vex}_{H_i^s} f_i(x_i) \right] + \text{vex}_{H_{i_t}^t} f_{i_t}(x_{i_t}).$$

Thus, if the underestimator over H^s is stored in memory, then the underestimator for H^t requires update only in one coordinate i_t. This is simply because of separability of the functions. The crucial thing, however, is whether $\text{vex}_{H_{i_t}^t} f_{i_t}$ needs to be computed from scratch using the breakpoints of f_{i_t} and employing Algorithm 4.1 for univariate PLFs. This is not necessary because of a property of PLFs that only a subset of the breakpoints of $\text{vex}_{H_{i_t}^t} f_{i_t}$ is different than those of $\text{vex}_{H_{i_t}^s} f_{i_t}$ as we show in Section 4.2. This allows for (on average) a quick and fast update to the underestimator of f_{i_t} over $H_{i_t}^s$ (assuming that it is stored in memory), although in the worst case, it is possible that all the breakpoints have to be updated. Hence, we calculate $\underline{v}(H^t)$ by starting with the relaxation (4) for $\underline{v}(H^s)$ and modifying some of the linear constraints in (4b) as needed for $i = i_t$ and $k \in \mathcal{E}_{i_t}(H^t)$. If S is a polyhedron, we can then employ the dual-simplex method to compute $\underline{v}(H^t)$ starting from $\underline{v}(H^s)$, which is generally significantly faster than using the primal simplex for $\underline{v}(H^t)$.

Remark 3.1. The sBB algorithm developed here can be modified to accommodate separable PLFs in constraints using similar arguments as the classical results by Soland (1971). Yet, for ease of exposition, we restrict our attention in this paper to PLFs in the objective only. Similarly, it is possible to integrate the methods developed here in general-purpose (spatial) branch-and-bound-based solvers and solve a broader class of mixed-integer nonlinear problems.

3.2. Relation to MILP and SOS2 Approaches

It is well known that all of the MILP models for PLFs share the sharpness property when the functions are l.s.c.; their LP relaxations (when S is a polyhedron) give the same lower bound as convexifying each function over its interval domain, which is equivalent to our relaxation (4). However, upon branching, most relaxations lose this guarantee of providing the same bound as (4). In fact, only the incremental and SOS2 models share this property called *hereditary sharpness* (Huchette and Vielma 2023). This property is very desirable because it leads to balanced search trees (Yıldız and Vielma 2013). Indeed, experiments indicate that the *incremental* and *SOS2* models perform very well on PLFs with a small number of segments and are only outperformed by the *logarithmic* model with growing numbers of segments (cf. Rebennack 2016, Huchette and Vielma 2023). These considerations have been summarized by Huchette and Vielma (2023, p. 1839) with the remark that "the high performance of the [logarithmic] formulation is due to its strength and small size and in spite of its poor branching behavior." The addition of some valid inequalities and cutting planes to the MILP model would strengthen the LP relaxation, but the fact remains that a desirable method for solving problems with PLFs should combine both hereditary sharpness and a small-scale formulation.

This gave the motivation to our sBB approach. By updating our convex underestimator over every subset H^k, we manually achieve relaxations of the strength (4) at every node of the branch-and-bound tree. Moreover, the LP relaxations are particularly small. In contrast to SOS2 and MILP formulations, the size of the relaxation (4) does not grow with the number of segments of f_i but with the number of segments of its envelope. To illustrate this, let K_i be the number of segments of f_i and E_i be the number of segments of $\text{vex}_{H_i} f_i$. If each f_i is continuous, the *logarithmic* MILP model adds $\sum_{i=1}^n K_i$ continuous variables, $\sum_{i=1}^n \lceil \log_2(K_i - 1) \rceil$ binary variables, and $\sum_{i=1}^n (2 \cdot \lceil \log_2(K_i - 1) \rceil + 3)$ constraints (cf. Vielma et al. 2010). In contrast, the sBB relaxation (4) adds n continuous variables, zero integers, and $\sum_{i=1}^n E_i$ constraints. Because $K_i \gg 1$ typically, we have far fewer variables. For the constraints, E_i is no more than K_i, although it can be more than $\log_2 K_i$. Hence, if the PLFs are such that their envelopes have few segments, then our relaxations will be smaller in size while being of the same strength as the conventional models. An extreme case of

this is when each f_i is concave where our relaxations will add n constraints, which can be much smaller than the number of constraints in MILP and SOS2 because of K_i being arbitrary.

Furthermore, the sBB offers the advantage of a sparser constraint matrix. Rebennack (2016) pointed out that formulations like the *logarithmic* model result in a dense constraint matrix. On the other hand, the sBB relaxation (4) is, in particular, sparse; each added inequality has exactly two nonzeros. Indeed, LPs with convex PLFs can be solved very efficiently by exploiting its structure (Fourer 1985, Gorissen 2022). Finally, spatial branching offers a higher degree of flexibility in branching decisions compared with integer or SOS2 branching. Both integer and spatial branching choose a branching variable x_i. However, although spatial branching can branch at any point in the interval $[l_i, u_i]$, integer branching in MILP and SOS2 models can be mapped to specific points in each interval. Therefore, spatial branching can mimic integer and SOS2 branching, but the converse is not true.

4. Convexifying Univariate PLFs

It was outlined in Section 3.1 that the key ingredient of this work is generation and efficient updates of convex continuous underestimators of univariate PLFs. Therefore, in this section, we focus on a univariate (possibly discontinuous) PLF $f : I = [l, u] \to \mathbb{R}$, where we omit the subscript i for ease of notation and better readability. The results derived here will be utilized in the sBB algorithm in the next section by applying them to the PLFs f_i in Problem (1).

Let f have $K + 1$ breakpoints in I for some integer $K \geqslant 1$, and these are indexed by the set $\mathcal{K} := \{0, 1, \ldots, K\}$, with the x values of the breakpoints being given by the set $B_f := \{b^k : k \in \mathcal{K}\}$, where $l = b^0 < b^1 < b^2 < \cdots < b^K = u$. The function values at the breakpoints are $y^k = f(b^k)$ for $k \in \mathcal{K}$. The left and right limits at each breakpoint are $y^{k,-}$ and $y^{k,+}$, respectively. For the left (respectively, right) endpoint, we set the left (respectively, right) limit to the function value. Thus, f is completely defined by the following finite collection of tuples as input:

$$\{(b^k, y^k, y^{k,-}, y^{k,+}) : k \in \mathcal{K}\}.$$

Note the following obvious fact.

Observation 4.1. Any finite set of points in $\mathbb{R}^2$ corresponds to a continuous univariate PLF obtained by doing linear interpolation between consecutive (taken w.r.t. x coordinates) points.

4.1. PLF Underestimator

We construct a tight convex and continuous PLF underestimator for f. To describe this, define the following PLF over I:

$$\underline{f}(x) := \begin{cases} \min\{y^k, y^{k,-}, y^{k,+}\}, & x = b^k \text{ for some } k \in \mathcal{K} \\ f(x), & x \in I \setminus B_f. \end{cases} \tag{5a}$$

Lemma 4.1. $\underline{f}$ is an l.s.c. PLF underestimator of f over I.

Proof. It is clear that $\underline{f}(x) \leqslant f(x)$ for all $x \in I$. It is continuous at $x \notin B_f$ because f is a PLF. At any breakpoint b^k, we have

$$\liminf_{x \to b^k} \underline{f}(x) = \min\left\{\lim_{x \uparrow b^k} \underline{f}(x), \lim_{x \downarrow b^k} \underline{f}(x)\right\} = \min\left\{\lim_{x \uparrow b^k} f(x), \lim_{x \downarrow b^k} f(x)\right\}$$
$$= \min\{y^{k,-}, y^{k,+}\} \geqslant \underline{f}(x),$$

and so, $\underline{f}$ is an l.s.c. function over I. □

But, this l.s.c. underestimator need not be convex. Hence, we convexify it to obtain the function

$$\mathrm{vex}_I f(x) := \mathrm{cvx}_I \underline{f}(x), \qquad x \in I, \tag{5b}$$

where cvx_I denotes the convex envelope operator over I. This underestimator has the following properties.

Proposition 4.1. $\mathrm{vex}_I f$ is a convex and continuous PLF underestimator of f whose breakpoints are given by the set

$$B_{\mathrm{vex}_I f} = \{l, u\} \bigcup \{b^k \in B_f : slope(i, k) < slope(j, k), \ \forall 0 \leqslant i < k < j \leqslant K\},$$

where slope$(i,k) := (\underline{f}(b^k) - \underline{f}(b^i))/(b^k - b^i)$ for all $i \neq k$. Furthermore, we have

$$\mathrm{vex}_I f(x) = \underline{f}(x), \qquad x \in B_{\mathrm{vex}_I f}.$$

We use the following technical results to establish the above claims on $\mathrm{vex}_I f$.

Lemma 4.2 (cf. Tuy 2016, Proposition 2.17). *A convex function is u.s.c. over any polyhedron P in its domain. Hence, if the function is l.s.c. over P, then it is actually continuous over P.*

Lemma 4.3. *A continuous univariate PLF is convex if and only if the slopes of its linear pieces form an increasing sequence when arranged from left to right.*

In the following condition from planar geometry, we say that three points form a *convex* (respectively, *concave*) *triangle* when the point in between lies below (respectively, above) the segment joining the other two points.

Lemma 4.4. *The continuous PLF formed by joining a finite set of points in $\mathbb{R}^2$ is a convex function if and only if every triplet of points forms a convex triangle. Consequently, if the PLF is nonconvex, then a point is not a breakpoint if and only if it forms a concave triangle with two other points, one to its left and one to its right.*

Proof. Necessity is obvious from the definition of convexity. Sufficiency can be argued by contraposition. Suppose that the PLF is not convex. We will use Lemma 4.3. Therefore, nonconvexity means that there exists some breakpoint x^i such that the slope to the left of x^i is greater than the slope to the right (equality of slopes is impossible because of x^i being a breakpoint). This implies that there is a nonconvex (concave) triangle with x^i as its apex. In particular, letting $x^i = \lambda x^{i-1} + (1-\lambda)x^{i+1}$ for some $\lambda \in (0,1)$, we have $\frac{y^i - y^{i-1}}{1-\lambda} > \frac{y^{i+1} - y^i}{\lambda}$, which after rearranging, becomes $y^i > \lambda y^{i-1} + (1-\lambda)y^{i+1}$, leading to a nonconvex triangle formed by the points indexed by $(i-1, i, i+1)$. $\square$

Proof of Proposition 4.1. Because $\mathrm{vex}_I f$ is the convex envelope of the PLF $\underline{f}$, it is obviously a convex PLF over I. Lemma 4.1 implies that this PLF is an underestimator of f. The convex envelope of an l.s.c. function is l.s.c. convex, is continuous over the interior of its domain, and can only be discontinuous on the boundary. Combining this fact with Lemma 4.2, where we use I being a polyhedron in $\mathbb{R}$, gives us that $\mathrm{vex}_I f$ is a convex and continuous underestimator.

The breakpoints of $\mathrm{vex}_I f$ must be breakpoints of $\underline{f}$ and hence, of f. The convex continuous PLF $\mathrm{vex}_I f$ is formed by joining its finitely many breakpoints. From Lemma 4.4, the characterization of the breakpoints of the underestimator follows immediately. The breakpoints of a PLF form what is more generally called the generating set in the global optimization literature for general nonconvex functions, and it is known that the envelope of an l.s.c. function equals the function value at points in its generating set. Hence, the underestimator equals $\underline{f}$ at its breakpoints. $\square$

Another convex underestimator to f is the convex envelope of f denoted by $\mathrm{cvx}_I f$. This equals $\underline{f}$ at its breakpoints in (l, u), whereas at the endpoints $\{l, u\}$, we may have inequality and so, can only say that $\mathrm{cvx}_I f(b^k) \geq \underline{f}(b^k)$ for $k \in \{0, K\}$. It is also not hard to see that $\mathrm{vex}_I f$ and $\mathrm{cvx}_I f$ have the same set of breakpoints. Therefore,

$$\mathrm{vex}_I f(x) = \mathrm{cvx}_I f(x), \quad x \in B_{\mathrm{vex}_I f} \setminus \{b^0, b^K\}, \qquad \mathrm{vex}_I f(x) \leq \mathrm{cvx}_I f(x), \quad x \in \{b^0, b^K\}. \tag{6}$$

Thus, the only difference between $\mathrm{vex}_I f$ and $\mathrm{cvx}_I f$ is in their values at the endpoints, where the latter will be u.s.c. because of Lemma 4.2 but may not be l.s.c.

We now build upon the characterization of breakpoints in Proposition 4.1 to derive an efficient algorithm for computing $\mathrm{vex}_I f$ given f as an input through its breakpoints.

Proposition 4.2. *Algorithm 4.1 produces $\mathrm{vex}_I f$ after $O(K)$ iterations.*

Proof. Each application of the while loop is repeatedly checking the necessary and sufficient conditions for the slopes from Lemma 4.4. Furthermore, because of the updates done to the lists where the last element is removed, at any stage the last two elements in the lists yield a lower bound on the slope required to make the kth point a breakpoint. This implies that the while loop executes only a constant number of times for each k, and so, the entire algorithm runs in $O(K)$ iterations. The points in the lists that it outputs indeed represent the breakpoints of $\mathrm{vex}_I f$ because they were obtained by checking the conditions in Lemma 4.4 and so, correspond to the characterization in Proposition 4.1. $\square$

Algorithm 4.1 (Generating a Convex Continuous Underestimator to a Discontinuous PLF)

> **Data:** Lists $B = \{b^0, b^1, \ldots, b^K\}$ and $Y = \{(y^k, y^{k,-}, y^{k,+}) : k \in \mathcal{K}\}$ of PLF $f : [l, u] \to \mathbb{R}$
>
> **Result:** Lists $\mathcal{B}$ and $\mathcal{Y}$ defining the tuples of $\mathrm{vex}_I f : [l, u] \to \mathbb{R}$.
>
> Compute $\overline{y}^k = \min\{y^k, y^{k,-}, y^{k,+}\}$ for $k = 0, 1, \ldots, K$
>
> Initialize $\mathcal{B} = \{b^0\}$ and $\mathcal{Y} = \{\overline{y}^0\}$
>
> **for** $k = 1$ *to* K **do**
>
>> **while** $|\mathcal{B}| \geqslant 2$ *and* $\frac{\mathcal{Y}[k] - \mathcal{Y}[-1]}{\mathcal{B}[k] - \mathcal{B}[-1]} < \frac{\mathcal{Y}[-1] - \mathcal{Y}[-2]}{\mathcal{B}[-1] - \mathcal{B}[-2]}$ **do**
>>
>>> Remove the last element of $\mathcal{B}$ and $\mathcal{Y}$.
>>
>> **end**
>>
>> Update $\mathcal{B} = \mathcal{B} \cup \{b^k\}$ and $\mathcal{Y} = \mathcal{Y} \cup \{\overline{y}^k\}$
>
> **end**
>
> **return** $\mathcal{B}$ and $\mathcal{Y}$

The worst-case running time of $O(K)$ for our algorithm cannot be improved further because a convex f would take K iterations because of every breakpoint of f also being a breakpoint of its envelope. However, it may be possible to improve the average running time by considering one of the many different algorithms in the literature; cf. Cormen et al. (2009, chapter 33.3) for generating the convex hull of a finite set of points in $\mathbb{R}^2$ (note that this convex hull is composed of the convex envelope, the concave envelope, and at most, two vertical segments). For example, the classical Graham's scan algorithm begins with a reference point having the smallest y coordinate, calculates the polar angles of the other points w.r.t. the reference point (equivalent to slopes of the line segments joining the two points), and then, applies Lemma 4.4 to discard points that will not be breakpoints of the envelope (Graham 1972). Our algorithm starts with the leftmost breakpoint as the reference point and compares slopes w.r.t. the previous candidate breakpoint. Although there are conceptual similarities with Graham's scan, it is not clear (and probably not true) that the two algorithms are in a bijection.

4.2. Updating Envelope over Subintervals

The branching procedure of sBB algorithms requires constant updating/recomputing of the underestimator $\mathrm{vex}_I f$ over a subinterval

$$I' := [\tilde{l}, \tilde{u}] \subset [l, u] = I.$$

Of course, Algorithm 4.1 can be used to compute $\mathrm{vex}_{I'} f$, but this would scan the breakpoints from scratch, which can be computationally expensive when there are many segments, and we show that this is not necessary. Yet, using Algorithm 4.1 to calculate $\mathrm{vex}_{I'} f$ requires rescanning all breakpoints of f in I'. Especially for PLFs with many segments, this can be an expensive computation. However, this is usually not necessary because we show that $\mathrm{vex}_{I'} f$ equals $\mathrm{vex}_I f$ over some part of I' in th middle and needs to be updated only over the end pieces. In particular, the envelope does not change between the leftmost and rightmost breakpoints in I', which can lead to substantial savings in computation if the subinterval is large w.r.t. I. To describe our result, let us denote

$$b^{lo} := \min\{b^k : b^k \in B_{\mathrm{vex}_I f} \cap [\tilde{l}, \tilde{u}]\}, \qquad b^{up} := \max\{b^k : b^k \in B_{\mathrm{vex}_I f} \cap (\tilde{l}, \tilde{u}]\}. \tag{7a}$$

Note that if b^{lo} and b^{up} do not exist, then the updated envelope is trivial. Henceforth, assume that they exist and partition I' into three intervals:

$$I^1 := [\tilde{l}, b^{lo}], \qquad I^2 := [b^{lo}, b^{up}], \qquad I^3 := [b^{up}, \tilde{u}]. \tag{7b}$$

Proposition 4.3. *Assume b^{lo} and b^{up} exist. The underestimator over I' can be described as follows:*

$$\mathrm{vex}_{I'} f(x) = \begin{cases} \mathrm{vex}_{I^1} \underline{f}(x), & x \in I^1 \\ \mathrm{vex}_I \underline{f}(x), & x \in I^2 \\ \mathrm{vex}_{I^3} \underline{f}(x), & x \in I^3. \end{cases}$$

Proof. The function on the right side of the equality is obtained by gluing together three different convex functions. Hence, we need to argue convexity of this glued function. But, this follows rather immediately from the

necessary and sufficient conditions in Lemmas 4.3 and 4.4. Because the breakpoints of f in I^1 were not breakpoints of $\text{vex}_I f$, they form a concave triangle with the breakpoints in I^2, and so, after convexifying over I^1, the slopes of the resulting linear segments can be no more than the slopes of the segments in I^2. Similar arguments hold for I^3. □

4.3. An Illustrative Example

The PLF in Figure 1 has five segments (so, $K = 5$) with the breakpoints $b^0 = 1, b^1 = 3, b^2 = 7, b^3 = 8, b^4 = 11, b^5 = 13$. Note that f is discontinuous at b^2 but otherwise, continuous. The tuples corresponding to the breakpoints are $(1,3)$, $(3,5)$, $(7,2,1,3)$, $(8,5)$, $(11,7)$, and $(13,7)$.

Applying Algorithm 4.1 to this function over $I = [1,13]$ receives as input the lists $B = [1,3,7,8,11,13]$ and $Y = [3,5,1,5,7,7]$ and outputs the lists $\mathcal{B} = [1,7,13]$ and $\mathcal{Y} = [3,1,7]$. They define the continuous PLF $\text{vex}_I f(x)$ formed by the tuples $(1,3)$, $(7,1)$, and $(13,7)$, which equals $\text{cvx}_I f(x)$ depicted in Figure 1 if Algorithm 4.1 is invoked to compute $\text{vex}_{I'} f$ over $I' = [7,13] \subset [1,13]$, and the input lists are $B = [7,8,11,13]$ and $Y = [2,5,7,7]$. Realize that $Y[0] = 2 \neq 1$ because the discontinuity at $b^0 = 7$ is at the edge of I', and hence, $\overline{y}^0 = \min\{2,2,7\}$ because $y^{0,-} = y^0$.

Let $I = [1,13]$ and $I' = [3,10]$. $\text{vex}_I f$ is given by $(1,3)$, $(7,1)$, and $(13,7)$. Hence, $b^{lo} = b^{up} = 7$. Consequently, $I^1 = [3,7]$, $I^2 = [7,7]$, and $I^3 = [7,10]$. φ is given by $(3,5)$, $(7,1)$, $(8,5)$, and $(10,6\frac{1}{3})$. Hence, $\text{vex}_{I^1}\varphi$ is formed by $(3,5)$ and $(7,1)$, and $\text{vex}_{I^3}\varphi$ is formed by $(7,1)$ and $(10,6\frac{1}{3})$. Finally, $\text{vex}_{I'} f$ is given by $(3,5)$, $(7,1)$, and $(10,6\frac{1}{3})$.

This example illustrates that Proposition 4.3 does not always lead to a reduction in the number of breakpoints to be scanned. However, if f is highly nonconvex with many segments, the savings can be enormous. Therefore, Proposition 4.3 is particularly useful for PLFs that accurately approximate a highly nonlinear function.

5. Spatial Branch-and-Bound Algorithm

Our main ideas for an sBB algorithm to solve the PLF optimization Problem (1) were sketched in Section 3.1. The algorithm is presented formally in Algorithm 5.1. The bounding operation is specified next, the branching schemes are in Section 5.2, and convergence is discussed in Section 5.3.

The following notation is used to describe our algorithm. Iteration number is k. For each k, H^k is the partition element; x^k and $\underline{v}(H^k)$ are the optimal solution and the optimal value of relaxation $\mathcal{R}^k$, respectively; α^k and β^k are the global upper and lower bounds, respectively, to v^*; and $\overline{x}^k$ is the incumbent solution. $\mathcal{L}$ denotes the list of unfathomed subproblems at any stage of the algorithm. The user-defined absolute termination gap is ε.

5.1. Node Relaxations

Each node of the search tree corresponds to a hyperrectangle $H^k \subseteq H$ and the subproblem

$$\mathcal{P}^k : \quad v(H^k) = \inf_x F(x) \quad \text{s.t.} \quad x \in S \cap H^k. \tag{8a}$$

Our lower bound on this nonconvex problem is denoted by $\underline{v}(H^k)$, which is obtained by solving the following convex relaxation:

$$\mathcal{R}^k : \quad v(H^k) \geqslant \underline{v}(H^k) = \min_x \text{vex}_{H^k} F(x) \quad \text{s.t.} \quad x \in S \cap H^k, \tag{8b}$$

Figure 1. (Color online) PLF $f(x)$ with Discontinuity at 7 as the Solid Line and Convex Envelope $\text{cvx}_I f(x)$ over Domain $I = [l,u] = [1,13]$ as the Dashed Line
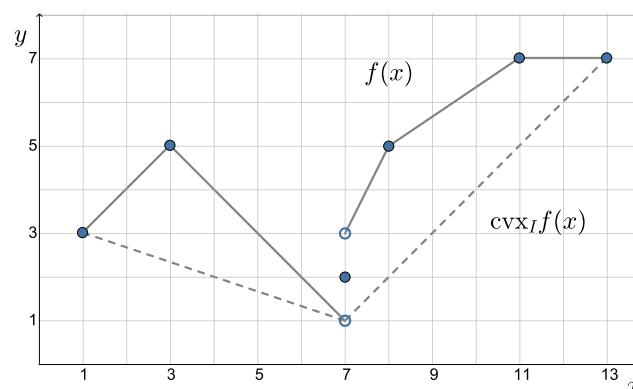
where the underestimator is defined as

$$\text{vex}_{H^k} F(x) = \sum_{i=1}^{n} \text{vex}_{H_i^k} f_i(x_i). \tag{8c}$$

Because $\text{vex}_{H^k} F$ is polyhedral as per the results of the previous section, using the epigraph modeling trick as in (4) leads to a tractable convex formulation for the node relaxation subproblem. It will be useful to separate a single coordinate from the above sum so that we can write

$$\text{vex}_{H^k} F(x) = \text{vex}_{H_j^k} f_j(x_j) + \sum_{i \neq j} \text{vex}_{H_i^k} f_i(x_i). \tag{8d}$$

In our context, the coordinate j will correspond to the branching variable that was used to create this node subproblem from its parent node in the sBB tree. In particular, if this node H^k was created from its parent node H^p by branching on x_{i_k}, then using $j = i_k$ in (8d) gives us

$$\text{vex}_{H^k} F(x) = \text{vex}_{H_{i_k}^k} f_{i_k}(x_{i_k}) + \sum_{i \neq i_k} \text{vex}_{H_i^p} f_i(x_i). \tag{8e}$$

Note that when using Proposition 4.3 to update the underestimator over a child node, the breakpoints of $\text{vex}_{H^k} F$ must be stored for each partition element H^k. It is common for sBB/B&B methods to store LP relaxation data in order to solve the child-node relaxation in a few iterations using the dual simplex rather than from scratch. However, if memory is scarce, Algorithm 4.1 can be called at each child node H^l to compute $\text{vex}_{H^l} F$ from scratch, and no additional data need to be stored.

Algorithm 5.1 (Spatial Branch-and-Bound Algorithm for PLF Optimization)

Root node: Compute $\text{vex}_H F$ as per (8c) using Algorithm 4.1 for $\text{vex}_{H_i} f_i$ for all i

Solve $\mathcal{R}^0$ to obtain x^0 and r^0

if $\mathcal{R}^0$ *is infeasible* **then** return $\mathcal{P}$ is infeasible

else Set $\mathcal{L} = \{H\}$, $k = 0$, $\alpha^0 = F(x^0)$, $\beta^0 = r^0$, and $\overline{x}^0 = x^0$

while $\mathcal{L} \neq \emptyset$ **do**

 Node selection: Find an $H^{l^*} \in \arg\min\{r^l : H^l \in \mathcal{L}\}$. Mark it as parent node, and set $H^k = H^{l^*}$ and $\beta^k = r^{l^*}$

 Branching: Partition H^k into $\mathcal{H} = \{H^{k,1}, H^{k,2}\}$ using a branching rule from Section 5.2. Let x_{i_k} denote the branching variable

 Bounding: for $l \in \{1, 2\}$ **do**

 Compute $\text{vex}_{H^{k,l}} F$ as per (8e) with $p = k$ and $k = k, l$ and using Proposition 4.3 to update the envelope in the coordinate i_k

 Solve relaxation $\mathcal{R}^l$ to obtain x^l and r^l

 if $\mathcal{R}^l$ *is infeasible* **then** remove $H^{k,l}$ from $\mathcal{H}$

 end

 Update: Set $k \leftarrow k + 1$. Examine whether the previous global upper bound α^{k-1} can be improved,

$$\alpha^k = \min\left\{\alpha^{k-1}, \min_{H^{k,l} \in \mathcal{H}} F(x^l)\right\}.$$

 Update the incumbent $\overline{x}^k$ accordingly.

 Add child nodes to list: $\mathcal{L} \leftarrow (\mathcal{L} \setminus \{H^k\}) \cup \mathcal{H}$

 Pruning: Fathom subproblems by bound dominance as $\mathcal{L} \leftarrow \mathcal{L} \setminus \{H^l : r^l \geqslant \alpha^k - \varepsilon\}$.

end

5.2. Branching Rules

Consider partition element H^k with the optimal solution x^k to its relaxation $\mathcal{R}^k$. We give three different rules for the *branching step* of Algorithm 5.1 to partition H^k into $H^{k,1}$ and $H^{k,2}$. The first follows the common concept to branch on the variable x_i, which causes the largest violation (i.e., contributes most to the convexification gap). It was first proposed by Falk and Soland (1969), and variations of it can be found, for instance, in the solver

BARON (Tawarmalani and Sahinidis 2004). It is similar to the integer branching rule, where the variable with the largest fractional part is chosen. The second branching rule follows the simple concept of branching at the midpoint of the longest edge and was used, for instance, in the solver αBB (Adjiman et al. 1998).

5.2.1. Largest-Error Branching Rule.
Select the index that contributes most to the convexification gap at x^k:

$$\tau \in \arg \max_{i=1,\ldots,n} [f_i(x_i^k) - \text{vex}_{H_i^k} f_i(x_i^k)], \tag{9a}$$

breaking ties using the smallest index rule. Partition H^k at the point x_τ^k:

$$H^{k,1} = \{x \in H^k : x_\tau \leqslant x_\tau^k\} \quad \text{and} \quad H^{k,2} = \{x \in H^k : x_\tau \geqslant x_\tau^k\}. \tag{9b}$$

5.2.2. Longest-Edge Branching Rule.
Select the index with the longest edge by

$$\tau \in \arg \max_{i=1,\ldots,n} u_i^k - l_i^k, \tag{10a}$$

breaking ties using the smallest index rule. Partition H^k at the midpoint of the longest edge:

$$H^{k,1} = \left\{x \in H^k : x_\tau \leqslant \frac{u_\tau^k + l_\tau^k}{2}\right\} \quad \text{and} \quad H^{k,2} = \left\{x \in H^k : x_\tau \geqslant \frac{u_\tau^k + l_\tau^k}{2}\right\}. \tag{10b}$$

5.2.3. Breakpoint Branching Rule.
Select the index τ by the largest-error rule (9a) applied only to breakpoints (i.e., select a breakpoint b_τ^* with the largest error). Partition H^k at this breakpoint:

$$H^{k,1} = \{x \in H^k : x_\tau \leqslant b_\tau^*\} \quad \text{and} \quad H^{k,2} = \{x \in H^k : x_\tau \geqslant b_\tau^*\}. \tag{11}$$

Preliminary computational experiments conducted on our test problems indicated a superiority of the *largest-error branching rule*. This computational superiority is also intuitive as this rule provides the maximum tightness at the former solution x^k for both child nodes, allowing for a visible increase in the lower bound and a balanced search tree. The other two branching rules do not possess these desirable computational properties, but they do have theoretical superiority because they allow for stronger convergence results as we explore in the next sections. We also note that integer branching applied to MILP-PLF models leads to unbalanced trees (cf. Yıldız and Vielma 2013).

5.3. Convergence Guarantees
Falk and Soland (1969, theorem 2) established asymptotic convergence of the largest-error branching rule when F is any continuous separable function. They also gave an example showing that for this rule, continuity of the functions is necessary for convergence. Under the weaker assumption of F being l.s.c., Falk and Soland (1969, theorem 1) established convergence under a stronger branching rule that creates more than two nodes at each step and thus, does not lead to binary search trees. Their results directly apply to our PLF optimization problem because we also consider a separable objective. Furthermore, as mentioned in Section 2.2 and described in Locatelli and Schoen (2013, chapter 5), finite convergence can also be obtained for general nonconvex optimization with $\varepsilon > 0$. However, we give some independent and self-contained proofs in this section. First, we show that the breakpoint rule yields finite convergence even with $\varepsilon = 0$.

Proposition 5.1. *When each f_i is l.s.c., Algorithm 5.1 using the breakpoint branching rule converges finitely for any $\varepsilon \geqslant 0$.*

Proof. The l.s.c. condition implies that $\underline{f_i}(x) = f_i(x)$ at a breakpoint $x \in B_{f_i}$, and so, our underestimator $\text{vex}_I f$ is exact at each breakpoint as per Proposition 4.1. Hence, a breakpoint is chosen at most once for branching because once it is branched upon, the underestimator will have zero error at this point throughout the subtree from this node. Because there are finitely many breakpoints, the claim follows because every feasible leaf node of the sBB tree will yield an exact representation of some restriction of the original Problem (1), and the union of all of these leaves will be Problem (1). $\square$

The largest-error rule is finitely convergent when $\varepsilon > 0$ and has asymptotic convergence when $\varepsilon = 0$. We give an independent proof of the second result by exploiting Lipschitz continuity of PLFs, which makes our arguments different than those of Falk and Soland (1969) for general separable functions.

Proposition 5.2. *When each f_i is continuous, Algorithm 5.1 using the largest-error branching rule converges in the limit for $\varepsilon = 0$.*

Proof. By construction, $\beta^k \leqslant v^* \leqslant \alpha^k$ for every k, and the sequence $\{\alpha^k\}$ is decreasing, whereas $\{\beta^k\}$ is increasing. Hence, if the sBB algorithm terminates at iteration p, we have $\alpha^p - \beta^p \leqslant \varepsilon$, and thus, the infimum v^* is found with ε-precision.

If the sBB algorithm does not terminate after a finite number of iterations, the sequence $\{H^k\}_k$ of partition elements is infinite. Thus, there must be at least one infinite nested subsequence of $\{H^k\}_{k\in\mathbb{N}}$ denoted by

$$\{H^q\}_{q\in Q} \quad \text{with} \quad H^{q+1} \subset H^q \quad \text{and} \quad Q \subseteq \mathbb{N}.$$

We have to show the consistent bounding property (i.e., there exists an infinite nested subsequence $\{H^q\}_{q\in Q}$ of $\{H^k\}_{k\in\mathbb{N}}$, for which $\lim_{q\to\infty}\alpha^q = \lim_{q\to\infty}\beta^q$). By boundedness of the sequences, we can extract subsequences such that $\{H^q\}_{q\in Q} \subset \{H^k\}_{k\in\mathbb{N}}$ with

   i. the sequence of optimal solutions x^q of relaxation $\mathcal{R}^q$ converges to a limit point x^+ and
   ii. only one index $\tau \in \mathcal{I}$ gets branched on infinitely often.

Because we are only interested in the limit behavior, we can, therefore, focus exclusively on the index τ. First, note that f_τ and thus, also $\text{vex}_{H_\tau^q} f_\tau$ are Lipschitz continuous with constant L_τ for all iterations q. Now, let us define function $\psi_\tau^q(x_\tau) = f_\tau(x_\tau) - \text{vex}_{H_\tau^q} f_\tau(x_\tau)$ over H_τ^q. Note that ψ_τ^q is Lipschitz with constant $2L_\tau$. By the largest-error branching rule, namely (9b), we obtain $x_\tau^{q-1} \in \text{bd}(H_\tau^q)$ and thus, $x_\tau^q, x_\tau^{q-1} \in H_\tau^q$. Consequently,

$$|\psi_\tau^q(x_\tau^q) - \psi_\tau^q(x_\tau^{q-1})| \leqslant 2L_\tau \cdot |x_\tau^q - x_\tau^{q-1}|.$$

Because f_τ is continuous and $x_\tau^{q-1} \in \text{bd}(H_\tau^q)$, we obtain that $\psi_\tau^q(x_\tau^{q-1}) = 0$, and hence,

$$|f_\tau(x_\tau^q) - \text{vex}_{H_\tau^q} f_\tau(x_\tau^q)| \leqslant 2L_\tau \cdot |x_\tau^q - x_\tau^{q-1}|.$$

Because $\lim_{q\to\infty} x_\tau^q = x_\tau^+$, we have that $\lim_{q\to\infty} |x_\tau^q - x_\tau^{q-1}| = 0$, and therefore,

$$\lim_{q\to\infty} |f_\tau(x_\tau^q) - \text{vex}_{H_\tau^q} f_\tau(x_\tau^q)| = 0.$$

Finally, there is a $\bar{q}$ so that for all $q > \bar{q}$, the branching index τ is selected by (9a). Hence, we get that

$$\forall i \in \mathcal{I} \setminus \{\tau\} : \lim_{q\to\infty}(f_i(x_i^q) - \text{vex}_{H_i^q} f_i(x_i^q)) = 0. \tag{12a}$$

Statement (i) follows then as a consequence of the definition of α^k, β^k, and H^q by

$$\lim_{q\to\infty}\alpha^q \leqslant \lim_{q\to\infty} F(x^q) = \lim_{q\to\infty} \text{vex}_{H^q} F(x^q) = \lim_{q\to\infty} r^q = \lim_{q\to\infty}\beta^q. \tag{12b}$$

For statement (ii), realize that f_τ has only finitely many breakpoints. Hence, after a finite iteration, $p \in Q$ holds that f_τ is affine over H_τ^p, and thus, $\psi_\tau^p(x_\tau^p) = f_\tau(x_\tau^p) - \text{vex}_{H_\tau^p} f_\tau(x_\tau^p) = 0$. By similar arguments, like in (12a) and (12b), it follows then that $\beta^p = \alpha^p$, and hence, $\{H^k\}_{k\in\mathbb{N}}$ is finite.

Now that consistent bounding has been established, convergence can be concluded by standard arguments from the literature (cf. Tuy and Horst 1988, theorem 2.3) (i.e., $\lim_{k\to\infty}\beta^k = v^* = \lim_{k\to\infty}\alpha^k$ and every accumulation point of $\{\bar{x}^k\}$ solves $\mathcal{P}$). Remember that $\{H^q\}_{q\in Q}$ is a subsequence of $\{H^k\}_{k\in\mathbb{N}}$, and thus, $\alpha^k = \alpha^q$ and $\beta^k = \beta^q$ for all $k = q \in Q$. By the monotony of the sequences $\{\beta^k\}$ and $\{\alpha^k\}$, convergence follows then directly by $\lim_{q\to\infty}\alpha^q = \lim_{q\to\infty}\beta^q$. $\square$

Wechsung and Barton (2014) imposed the requirement of *strongly consistent* on the branching scheme to obtain asymptotic convergence for general l.s.c. functions with the longest-edge branching rule. Their underestimators applied to PLFs are possibly no stronger than ours; so, their convergence result might carry over to our sBB for l.s.c. PLFs, but a rigorous exploration of this is left for future research.

6. Computational Experiments
6.1. Design of Experiments
We compare the computational performance of the sBB algorithm with MILP approaches from the literature as well as the state-of-the-art solver Gurobi. In Section 6.2, we consider continuous PLFs in network flow problems with concave PLFs (Section 6.2.1) and knapsack problems with both nonconcave and concave PLFs (Section 6.2.2). Discontinuous l.s.c. PLFs are tested for a network flow problem with fixed charges in Section 6.3. Further, we test

our sBB algorithm against the global solver Gurobi in Section 6.4. We conclude with a general discussion of our numerical results in Section 6.5.

Let us begin by outlining the design of our experiments. Algorithm 5.1 was implemented in Python version 3.11.9. The largest-error branching rule is chosen because in our initial testing, it seemed to do better than the other rules described in Section 5.2. Nodes were selected using the best-bound rule. The LPs on the nodes are solved with Gurobi. The MILP models are generated in Julia version 1.10 using the package PiecewiseLinearOpt developed by Huchette and Vielma (2023) and are solved by Gurobi. We use Gurobi version 11.0.3 with standard settings. All tests were carried out on a workstation with 4.70 GHz and 128 GB RAM running Windows 11 Enterprise. For termination, we used a relative optimality gap of 10^{-5} and a time limit of 30 minutes. All times given are wall-clock times. The code of the sBB implementation and the MILP generation as well as the instance generator are available at GitHub (Hübner et al. 2025).

We compare our sBB algorithm (sBB) against the PLF solver inside Gurobi (GRB) and four state-of-the-art logarithmic-sized MILP models available in the package PiecewiseLinearOpt. In particular, these are the *logarithmic* (Log) and *disaggregated logarithmic* (DLog) (Vielma et al. 2010) as well as the recently introduced *binary zigzag* (ZZB) and *general integer zigzag* (ZZI) models (Huchette and Vielma 2023). In contrast to these four logarithmically sized MILP formulations, to our knowledge, Gurobi's PLF solver is built on a linear-sized MILP model.

Similar to findings in the literature (cf. Vielma et al. 2010), first experiments indicated that linear-sized MILP models are not competitive to logarithmic-sized models when nonconvex PLFs with 50 or more segments are involved. Therefore, we restrict our comparisons to the four logarithmic-sized MILP models above available in the literature.

6.2. Continuous PLFs

6.2.1. Network Flow Problem with Concave Cost.
Network flow problems with nonconvex PLFs occur in many applications ranging from telecommunications to logistics (Croxton et al. 2007). They can be defined as follows:

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}(x_{ij}) \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = d_i \quad i = 1, \ldots, n \\
& l_{ij} \leqslant x_{ij} \leqslant u_{ij} \quad i, j = 1, \ldots, n .
\end{aligned}
$$

An instance of the network flow problem is created similar to Keha et al. (2006), Vielma et al. (2010), and Huchette and Vielma (2023) as follows. First, declare each node $i = 1, \ldots, n-1$ a demand, supply, or transshipment node with equal probability $\frac{1}{3}$. The transshipment nodes have $d_i = 0$, whereas the demand and supply nodes have $d_i \sim \pm\text{Uniform}(5, 50)$. To obtain a balanced problem, the final node n has $d_n = -\sum_{i=1}^{n-1} d_i$. The breakpoints $(b_i^k, f(b_i^k))$, $k = 0, \ldots, K$ of the concave PLFs $f_i(x_i)$ are determined as follows. Set $b_i^0 = l_i = 0$ and $b_i^K = u_i \sim \text{Uniform}(5, 50)$; generate $K-1$ points $b_i^k \sim \text{Uniform}(l_i, u_i)$, $k = 1, \ldots, K-1$; and order them. Subsequently, generate K slopes by $\text{slopes}_k \sim \text{Uniform}(1, 2000)/1000$, $k = 1, \ldots, K$, and order them in decreasing order to obtain a concave PLF. Finally, set $f_i(b_i^0) = 0$, and compute the y coordinates of the breakpoints by $f_i(b_i^k) = \text{slope}_k \cdot (b_i^k - b_i^{k-1}) + f_i(b_i^{k-1})$, $k = 1, \ldots, K$.

We perform our computational test on network flow problems with $n = 10$ nodes. For each K, 50 random network flow instances are generated and solved. The statistics of the solve times are given in Table 1. We display the median, the arithmetic mean, and the standard deviation as well as the number of instances that cannot be solved by a method within the time limit (fail) and the number of instances in which each method was the fastest (win).

6.2.2. Knapsack Problem with Approximated Nonlinearities.
As discussed in Section 1, PLFs are often used to approximate difficult nonlinear expressions in optimization problems. To test the sBB and MILP methods in this context, we consider the following nonlinear continuous knapsack problem:

$$
\min \sum_{i=1}^{n} f_i(x_i) \quad \text{s.t.} \quad \sum_{i=1}^{n} x_i = d, \quad l_i \leqslant x_i \leqslant u_i, \ i = 1, \ldots, n .
$$

Each $f_i(x_i)$ is a nonconvex continuous PLF randomly generated by approximating a smooth nonconvex function from Table 2. The functions therein are mostly taken from Casado et al. (2003).

6.2.2.1. Nonconvex, Nonconcave Knapsack Problems.
A random instance of the knapsack problem is then generated as follows. First, n functions h_i with bounds l_i and u_i are arbitrarily drawn from Table 2. Second, $K-1$

Table 1. Solve Times (Seconds) for Network Flow Problems with Continuous Concave PLFs

Method	Med.	Avg.	Std.	Win	Fail
Panel A: 10 segments					
ZZI	**0.26**	0.28	0.12	20	0
Log	**0.26**	0.34	0.50	12	0
DLog	**0.33**	0.32	0.14	9	0
ZZB	**0.39**	0.41	0.22	4	0
GRB	**0.40**	0.36	0.14	5	0
sBB	**4.55**	8.96	14.65	0	0
Panel B: 100 segments					
ZZI	**1.82**	2.11	1.06	24	0
ZZB	**1.95**	2.30	1.15	15	0
Log	**2.36**	2.68	1.19	5	0
DLog	**3.90**	4.67	2.20	0	0
sBB	**4.11**	6.17	7.08	6	0
GRB	**9.38**	9.65	3.95	0	0
Panel C: 500 segments					
sBB	**8.2**	11.1	11.2	39	0
Log	**15.1**	17.2	8.0	6	0
ZZI	**15.5**	18.0	9.0	4	0
ZZB	**15.8**	19.4	11.9	1	0
DLog	**23.6**	27.6	14.6	0	0
GRB	**90.0**	114.6	94.8	0	0
Panel D: 1,000 segments					
sBB	**5.8**	10.9	14.6	50	0
Log	**45.7**	49.5	18.2	0	0
DLog	**46.5**	57.4	32.4	0	0
ZZI	**48.4**	48.9	21.1	0	0
ZZB	**61.6**	61.2	23.9	0	0
GRB	**270.6**	363.4	241.9	0	0
Panel E: 5,000 segments					
sBB	**7.2**	11.0	11.0	50	0
Log	**330**	331	132	0	0
ZZI	**333**	329	152	0	0
DLog	**379**	405	200	0	0
ZZB	**515**	518	198	0	0
GRB	**1,800**	1,800	0	0	50
Panel F: 10,000 segments					
sBB	**8**	12	15	50	0
Log	**729**	763	294	0	0
DLog	**940**	876	374	0	1
ZZI	**976**	924	299	0	1
ZZB	**1,419**	1,368	410	0	9
GRB	**1,800**	1,800	0	0	50

Notes. sBB is the proposed method in this paper. Avg., arithmetic mean; Fail, number of instances that cannot be solved by a method within the time limit; Med., median; Std., standard deviation; Win, number of instances in which each method was the fastest. The methods are sorted according to the bold numbers.

points $b_i^k \sim \text{Uniform}(l_i, u_i), k \neq \{0, K\}$ are generated and ordered. The first and last breakpoints are set to $b_i^0 = l_i$ and $b_i^K = u_i$, respectively. Each h_i is then approximated by a PLF f_i with K segments given by the breakpoints $(b_i^k, h_i(b_i^k))$. The demand parameter d is then as well randomly determined by $d \sim \text{Uniform}(l + \frac{1}{4} \cdot (u - l), u - \frac{1}{4} \cdot (u - l))$, in which $l = \sum_{i=1}^n l_i$ and $u = \sum_{i=1}^n u_i$. We perform our computational test on knapsack problems of dimension $n = 100$. For each K, 50 random knapsack instances are generated and solved. The statistics of the solve times are given in Table 3.

In addition, we are interested in the impact of more segments on the approximation quality. Thereby, a knapsack problem is generated like described above, and each function h_i is approximated by a PLF f_i, which has $K + 1$ equidistantly distributed breakpoints. Then, the piecewise linear optimization problem is solved with solution x^K. The real objective value of the nonlinear problem given this point is $v^K = \sum_i h_i(x_i^K)$. Table 4 shows the relative

Table 2. Nonconvex Univariate Functions

No.	Function	Domain
1	$e^{-3x-12} - x^2 + 20$	$[-5, 5]$
2	$-0.2 \cdot e^{-x} + x^2$	$[-5, 5]$
3	$x^3 \cdot e^{-x^2}$	$[-5, 5]$
4	$\frac{x^5 - 20x^2 + 5}{x^4 + 1}$	$[-10, 10]$
5	$\log(3x) \cdot \log(2x) - 1$	$[0.1, 10]$
6	$10 \log(x) - 3x + (x - 5)^2$	$[0.1, 10]$
7	$\frac{-x^5 - 10x^2}{x^6 + 5}$	$[-10, 10]$
8	$x \cdot e^{-x^2}$	$[-5, 5]$
9	$-\frac{x^7}{5040} + \frac{x^5}{120} - \frac{x^3}{3} + x$	$[-4, 4]$
10	$\frac{x^2 - 5x + 6}{x^2 + 1} - 1$	$[-10, 10]$
11	$x^4 - 12x^3 + 47x^2 - 60x$	$[-1, 7]$
12	$x^6 - 15x^4 + 27x^2 + 250$	$[-4, 4]$
13	$x^4 - 10x^3 + 35x^2 - 50x + 24$	$[0, 5]$
14	$0.2x^5 - 1.25x^4 + 2.33x^3 - 2.5x^2 + 6x$	$[-1, 4]$
15	$x^3 - 7x + 7$	$[-4, 4]$
16	$\frac{(x^4 - 4x + 10)}{(x^2 + 1)} - 1$	$[-5, 5]$
17	$-x^5 \cdot e^{-x^2}$	$[-10, 10]$
18	$x^5 - 3x^4 + 4x^3 + 2x^2 - 10x - 4$	$[-1.5, 3]$
19	$\frac{(x^3 - 5x + 6)}{(x^2 + 1)} - 1$	$[-5, 5]$
20	$\frac{1}{x} + 2 \log(x) - 2$	$[0.1, 10]$

improvement in the real objective value if the approximation is refined (i.e., the value $-(v^{K+1} - v^K)/|v^K|$, where $K + 1$ means the next K value in Table 4 (e.g., $K = 20$ and $K + 1 = 50$)).

6.2.2.2. Concave Knapsack Problems. To evaluate the impact of nonconcavity on the solution methods, we also solve instances of knapsack problems where the PLFs are concave. Results are presented in Table 5. The knapsack problems are generated as before. To obtain a concave PLF, the slopes of the segments are computed and sorted in decreasing order. Subsequently, the y value of each breakpoint is recomputed by using the new slopes and x coordinate of the breakpoints. Table 5 shows that problems with concave PLFs are in general harder to solve for every method than problems with nonconcave PLFs. Indeed, nonconcave PLFs have at least one more convex segment than concave PLFs, which allows for tighter lower bounds.

6.2.3. Details on Computational Experiments. This section dives deeper into our numerical results. Means and medians are point estimators that do not necessarily provide a complete picture of the algorithms' performance on the randomly generated data set, and means can be distorted by heavy outliers. Therefore, in addition to the statistics provided in the preceding tables, we further investigate the behavior of the different models and algorithms by plotting the performance profiles of their solution times. We also investigate the amount of time that the sBB spends on its different operations.

6.2.3.1. Performance Profiles. Each model/algorithm gets one profile curve, which is interpreted as its approximate cumulative distribution function. This implies that the curves in panel (a) of Figure 2 and panel (a) of Figure 3 have stochastic dominance over other curves and hence, correspond to the best method. The horizontal axes in Figures 2 and 3 are relative running times obtained by dividing by the shortest running time. The vertical intercepts in Figures 2 and 3 give the number of instances for which each method solved the fastest (the win column in the associated tables).

Figures 2 and 3 give these profiles, respectively, for the network flow problems and knapsack problems with concave PLFs. In the former, the sBB profiles are consistent with Table 1 and give superior performance for

Table 3. Solve Times (Seconds) for Nonconcave Knapsack Problems

Method	Med.	Avg.	Std.	Win	Fail
	Panel A: 10 segments				
GRB	**0.03**	0.04	0.02	36	0
Log	**0.04**	0.12	0.50	7	0
ZZI	**0.05**	0.05	0.02	6	0
DLog	**0.05**	0.06	0.04	1	0
ZZB	**0.06**	0.07	0.03	0	0
sBB	**0.15**	0.24	0.27	0	0
	Panel B: 100 segments				
Log	**0.50**	0.53	0.23	44	0
ZZI	**0.72**	0.77	0.40	5	0
DLog	**0.82**	1.08	1.15	0	0
ZZB	**0.82**	0.87	0.47	1	0
sBB	**1.05**	1.75	2.33	0	0
GRB	**1.05**	1.19	0.66	0	0
	Panel C: 500 segments				
Log	**2.5**	3.0	1.7	41	0
sBB	**4.0**	9.8	23.1	4	0
ZZI	**4.2**	5.3	4.5	3	0
ZZB	**5.0**	6.3	7.4	1	0
DLog	**5.9**	9.2	11.0	1	0
GRB	**24.8**	486.9	786.5	0	13
	Panel D: 1,000 segments				
sBB	**7.0**	18.3	33.8	13	0
Log	**7.6**	8.5	7.2	34	0
DLog	**15.9**	22.9	23.6	2	0
ZZI	**18.1**	18.1	12.1	1	0
ZZB	**19.7**	20.1	14.2	0	0
GRB	**1,800**	962.6	880.6	0	26
	Panel E: 5,000 segments				
sBB	**58.8**	100.0	98.5	25	0
Log	**71.8**	103.1	86.5	24	0
DLog	**149.4**	331.3	380.9	0	1
ZZI	**188.9**	213.3	131.9	1	0
ZZB	**197.8**	240.7	202.2	0	0
GRB	**1,800**	1,800	0	0	50
	Panel F: 10,000 segments				
sBB	**111**	229	359	31	2
Log	**208**	470	550	17	5
DLog	**327**	549	543	2	5
ZZB	**462**	649	521	0	6
ZZI	**487**	611	484	0	4
GRB	**1,800**	1,800	0	0	50

Notes. sBB is the proposed method in this paper. Avg., arithmetic mean; Fail, number of instances that cannot be solved by a method within the time limit; Med., median; Std., standard deviation; Win, number of instances in which each method was the fastest. The methods are sorted according to the bold numbers.

Table 4. Relative Improvement in Real Objective Value over Previous Numbers of Segments K

K	Min.	Med.	Avg.	Max.	Std.
20	−486.67%	25.87%	**39.00%**	347.59%	100.62%
50	−3.37%	5.32%	**7.78%**	37.73%	7.81%
100	−0.87%	1.33%	**1.84%**	9.77%	1.82%
500	0.072‰	5.974‰	**7.646‰**	21.176‰	5.246‰
1,000	−0.197‰	0.205‰	**0.302‰**	2.060‰	0.415‰
5,000	0.011‰	0.107‰	**0.168‰**	0.839‰	0.173‰
10,000	−0.001‰	0.003‰	**0.004‰**	0.027‰	0.005‰

Notes. For $K = 20$, the improvement in real objective value is measured relative to the value of $K = 10$. Min., the minimum; Med., median; Avg., arithmetic mean; Max., the maximum; Std., standard deviation. The methods are sorted according to the bold numbers.

**Table 5.** Solve Times (Seconds) for Concave Knapsack Problems

Method	Med.	Avg.	Std.	Win	Fail
Panel A: 10 segments					
Log	**0.04**	0.05	0.02	26	0
GRB	**0.04**	0.06	0.03	15	0
DLog	**0.05**	0.06	0.02	1	0
ZZI	**0.05**	0.06	0.02	8	0
ZZB	**0.05**	0.06	0.03	0	0
sBB	**0.12**	0.19	0.19	0	0
Panel B: 100 segments					
Log	**0.70**	0.73	0.33	27	0
ZZI	**1.00**	1.06	0.71	3	0
DLog	**1.07**	1.05	0.48	1	0
sBB	**1.12**	2.42	4.56	19	0
ZZB	**1.12**	1.11	0.58	0	0
GRB	**3.19**	3.35	1.70	0	0
Panel C: 500 segments					
sBB	**2.3**	33.4	83.3	32	0
Log	**5.0**	8.4	10.6	16	0
ZZI	**8.6**	19.9	33.3	0	0
ZZB	**8.8**	20.1	30.5	0	0
DLog	**10.6**	25.0	57.6	2	0
GRB	**164.4**	621.8	743.0	0	13
Panel D: 1,000 segments					
sBB	**2.6**	95.2	309.9	37	1
Log	**10.8**	28.3	41.4	12	0
DLog	**21.7**	74.0	253.2	1	1
ZZI	**31.8**	102.9	275.5	0	1
ZZB	**36.0**	109.3	279.4	0	1
GRB	**1,800**	1,220	731.9	0	29
Panel E: 5,000 segments					
sBB	**22.4**	390.3	658.5	40	8
Log	**97.2**	584.1	735.2	2	10
DLog	**147.1**	719.7	782.6	0	16
ZZB	**580.5**	948.6	741.0	0	19
ZZI	**714.2**	948.6	681.2	0	17
GRB	**1,800**	1,800	0.0	0	50
Panel F: 10,000 segments					
sBB	**9**	300	604	45	5
Log	**215**	666	697	0	12
DLog	**401**	786	700	0	14
ZZB	**1,518**	1,269	574	0	24
ZZI	**1,571**	1,288	593	0	24
GRB	**1,800**	1,800	0	0	50

Notes. sBB is the proposed method in this paper. Avg., arithmetic mean; Fail, number of instances that cannot be solved by a method within the time limit; Med., median; Std., standard deviation; Win, number of instances in which each method was the fastest. The methods are sorted according to the bold numbers.

500 segments and beyond. The profiles for the concave knapsacks reveal that for up to 1,000 segments, the actual performance of sBB is much better than the high values for average times in Table 5. At 100 segments, sBB is quickest on the same number of instances as Log and dominates DLog and the two zigzag models, whereas beyond 500 segments, the dominance of sBB keeps growing steadily. Similar behavior is observed for the non-concave PLFs, and so, their profiles are omitted. This underscores the point that the average numbers in Table 5 are a bit distorted and do not provide complete information on performance of the algorithms.

6.2.3.2. Timing Statistics for the sBB. Here, we take a look at some details of the operation of the sBB implementation. Table 6 indicates that solving LPs takes only a small share of the sBBs solution time, although it is by far the most complicated operation in a branch-and-bound algorithm. Instead, operations like building the model and repeatedly adding constraints over the Python-Gurobi interface, evaluating PLFs, and generating the

Figure 2. (Color online) Performance Profiles for Network Flow Problems with Continuous Concave PLFs from Table 1
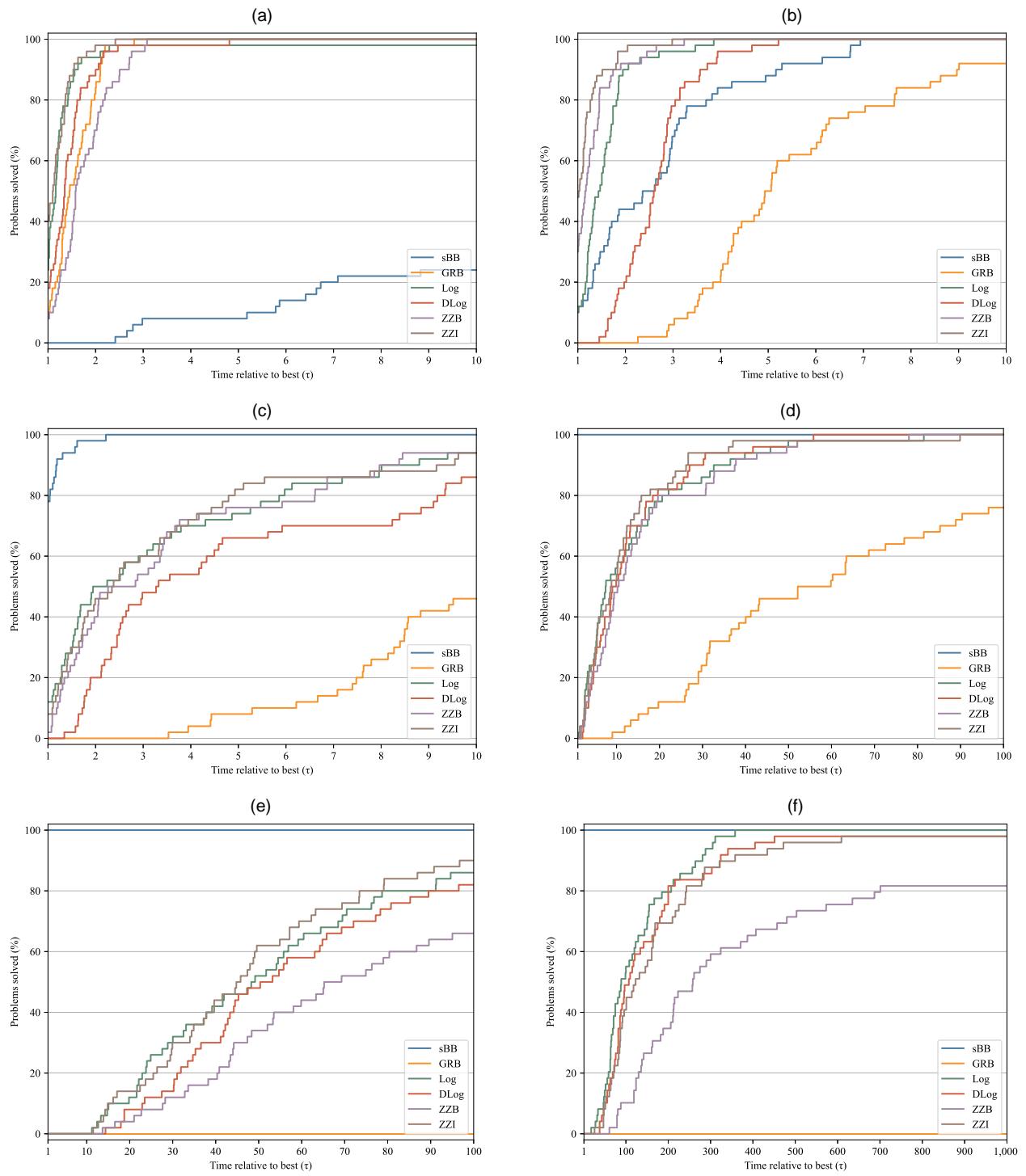
Notes. (a) Ten segments. (b) One hundred segments. (c) Five hundred segments. (d) One thousand segments. (e) Five thousand segments. (f) Ten thousand segments.

envelope take a high share. This is another indicator that an integration into a fully developed solver, such as Gurobi or BARON, would result in considerable speedups.

In addition, Table 6 indicates that the generation of the convex envelope takes more time if the PLF is concave. The reason for that is the while loop of Algorithm 4.1, which is always entered because every point results in a concave turn. However, if it is a priori known that the PLF is concave, then one could modify the algorithm to make it simply output the first and last breakpoints of the PLF without entering any loop.

Figure 3. (Color online) Performance Profiles for Concave Knapsack Problems from Table 5
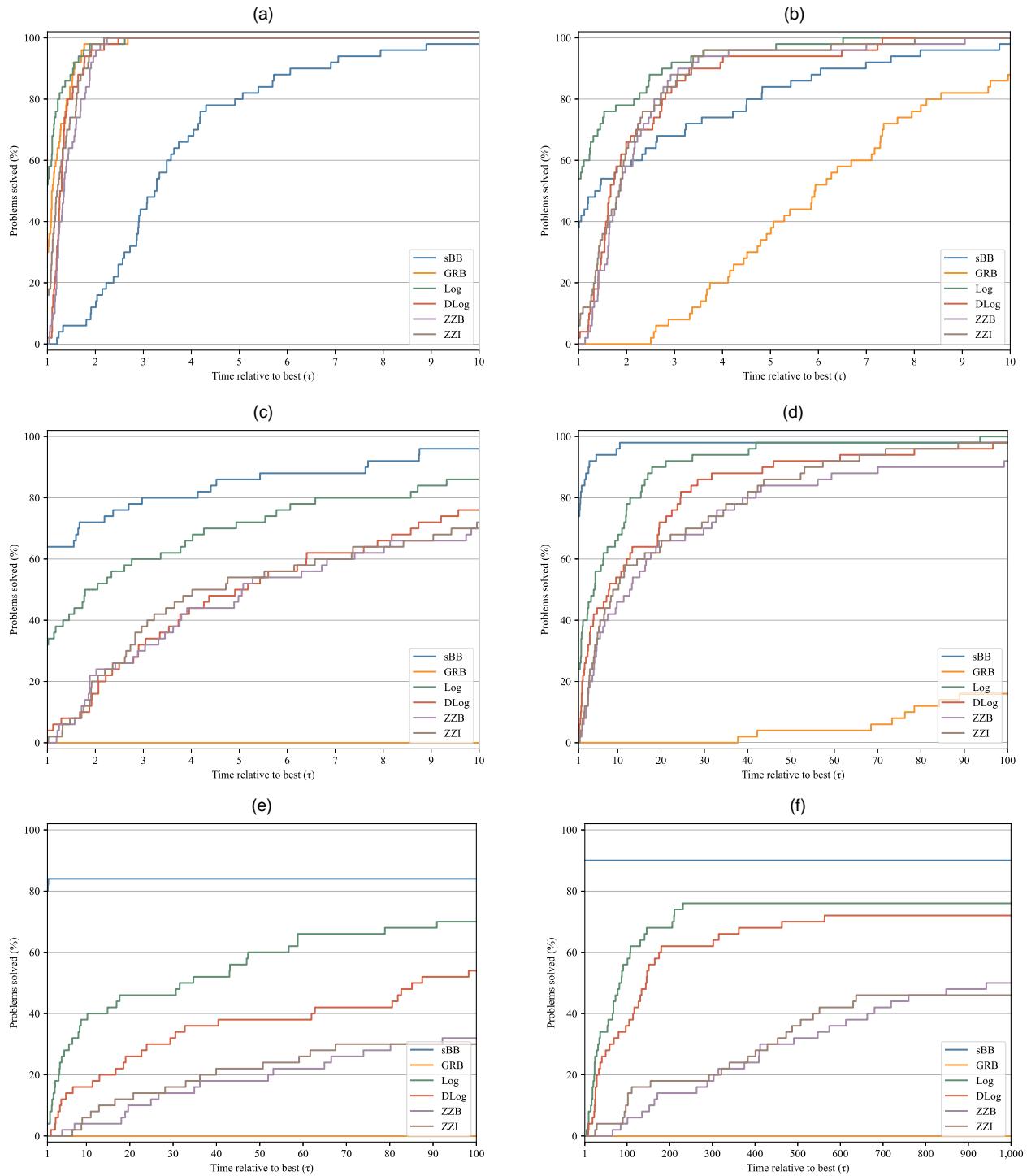
Notes. (a) Ten segments. (b) One hundred segments. (c) Five hundred segments. (d) One thousand segments. (e) Five thousand segments. (f) Ten thousand segments.

6.3. Discontinuous l.s.c. PLF

In many real-world applications in logistics, supply chains, and telecommunications, the network flow problem involves *fixed charges* (Rebennack et al. 2009). Those are fixed costs that are incurred as soon as a flow f_{ij} is strictly positive ($f_{ij} > 0$). They can represent real-world setup costs, like opening shipping lanes or starting equipment. However, they turn the continuous concave piecewise linear cost function of Section 6.2.1 into a *discontinuous* but lower semicontinuous PLF.

Table 6. Average Proportion of Run Time That Is Allotted to the Various Suboperations of the sBB Algorithm When Solving Knapsack Problems

	Nonconcave, %		Concave, %	
Operation	$K = 10$	$K = 10,000$	$K = 10$	$K = 10,000$
Gurobi interface	68	76	33	10
Solving LPs	2	15	2	1
Envelope generation	0	1	0	35
PLF evaluations	28	1	62	49
Other operations	2	8	3	4

To test the sBB algorithm under this discontinuous setting, we generate the network flow problem as described in Section 6.2.1 but add to every cost function a fixed-charge jump at $f_{ij} = 0$ given by a random uniformly distributed number between 10 and 50. We compare the sBB with the largest-error branching rule against the built-in PLF solver of Gurobi, which can also handle discontinuous l.s.c. PLFs. The logarithmic formulations either do not support discontinuous PLFs or are not implemented in the package PiecewiseLinearOpt (Huchette and Vielma 2023). The results are displayed for 50 random instances in Table 7.

Note that the sBB with the largest-error branching rule has no asymptotic convergence guarantee in general. Falk and Soland (1969) present an example that showcases a corner case where the sBB never converges. However, in our experiments, the sBB with the largest-error branching rule always converged. In Section 5.3, we pointed out that the breakpoint branching rule could achieve finite convergence even for discontinuous PLFs. However, we did not fully implement the breakpoint branching rule as early experiments indicated poor performance. This poor performance is caused by the failure to provide good improvements in the lower bounds after branching. The breakpoint branching rule does not necessarily branch in the surrounding of the solution of the parent node—in contrast to the largest-error rule—and thus, cannot guarantee a tighter convex envelope around the parent's solution after branching. Consequently, the parent's optimal point might also be the optimal point of the child node, and no improvement in the lower bound is gained. This branching behavior is similar to that of integer PLF branching rules and leads to imbalanced search trees (Yıldız and Vielma 2013). This sharply contrasts the largest-error branching rule, which branches directly at the parent's solution, thus guaranteeing an increase in lower bounds and a balanced search tree. The following example illustrates this.

Table 7. Solve Times (Seconds) for Network Flow Problems with Fixed Charges (Discontinuous PLFs)

Method	Med.	Avg.	Std.	Win	Fail
	Panel A: 10 segments				
GRB	**0.76**	0.78	0.41	50	0
sBB	**18.52**	59.92	98.25	0	0
	Panel B: 100 segments				
GRB	**16.63**	17.18	8.86	34	0
sBB	**24.29**	71.05	111	16	0
	Panel C: 500 segments				
sBB	**17.2**	84.5	159.5	46	0
GRB	**90.0**	147.6	183.4	4	0
	Panel D: 1,000 segments				
sBB	**22.0**	61.6	96.3	50	0
GRB	**291.2**	599.9	629	0	5
	Panel E: 5,000 segments				
sBB	**25.8**	110.8	232.8	50	0
GRB	**1,800**	1,800	0	0	50
	Panel F: 10,000 segments				
sBB	**33**	92	134	50	0
GRB	**1,800**	1,800	0	0	50

Notes. sBB is the proposed method in this paper. Avg., arithmetic mean; Fail, number of instances that cannot be solved by a method within the time limit; Med., median; Std., standard deviation; Win, number of instances in which each method was the fastest. The methods are sorted according to the bold numbers.

Example 6.1. Consider the problem

$$\min f_1(x_1) + f_2(x_2) \quad \text{s.t.} \quad x_1 + x_2 \geqslant 1, \quad x_1, x_2 \in [0, 2]$$

with continuous PLFs f_1 and f_2 given by the three breakpoints (0,0), (1,10), and (2,15) as well as (0,0), (1,2), and (2,1), respectively.

It is easy to verify that the optimal solution of the above problem is (0,2) with an optimal value of one. Given the convex envelopes of both PLFs, the root-node solution would be (0,1). The breakpoint branching rule would then branch at the breakpoint (1,10) of f_1. The convex envelopes of f_1 and f_2 would thus not be tightened around (0,1). Point (0,1) would still be a solution for the left child node, leading to no improvement of the lower bound and an imbalanced search tree. This contrasts the largest-error branching rule, which would branch at (0,1), tightening the convex envelope of f_2 at (0,1) and ensuring that the solution is found in the next iteration.

By adding breakpoints to function f_1, it would not be difficult to extend the above example so that the breakpoint branching rule would continue to branch on the unimportant variable x_1 for any number in $\mathbb{N}$. The breakpoint branching rule would not be able to detect that finding the optimal solution would require a single branching on variable x_2.

We believe that when designing convergent and computationally efficient branching rules for discontinuous PLFs, the idea of branching around the previous solution should be the guiding star as only that guarantees balanced search trees, which are essential for efficient B&B algorithms. However, the design of tailored branching rules for discontinuous PLFs, which have both theoretical convergence guarantees and are computationally efficient, is out of the scope of this work and is left for future research.

6.4. Comparison with Global MINLP Solvers

As mentioned before, to motivate this work, PLFs can be used to approximate nonlinear functions within mixed-integer nonlinear program (MINLP) problems to yield MILP problems (Füllner and Rebennack 2022). Therefore, we want to compare our proposed sBB with a global solver on some nonconvex nonlinear optimization problems. However, the results of this comparison need to be interpreted carefully as global solvers guarantee global optimality of the computed solutions—if they converge and the assumptions of the underlying algorithms are met—whereas our tested sBB method uses a static a priori approximation of the problem with 10,000 segments. Nevertheless, such a comparison can give insights into the scalability of our sBB versus the global solver tested.

The most well-known global solvers are ANTIGONE, BARON, SCIP, and LindoGLOBAL. These are all based on an sBB algorithm that computes the lower bound by disaggregating functions into elementary functions, such as $\log(x)$, a polynomial, or a bilinear function $x \cdot y$. To compute lower bounds, those elementary functions are replaced by a known convex underestimator. For more details on global solvers, we refer to Burer and Letchford (2012). Since its recent release of version 11, Gurobi also provides a global solver. This global solver is also based on an sBB using disaggregation into elementary functions. See the documentation on the website of Gurobi (Gurobi Optimization 2024) for more details.

Global solvers disaggregate more complex functions, such as those in Table 2, into a cascade of supported univariate functions (Burer and Letchford 2012). Solvers such as ANTIGONE, BARON, SCIP, and LindoGLOBAL do this behind the scenes. However, in the current version of Gurobi, the user needs to disaggregate this manually (Gurobi Optimization 2024). Either way, the disaggregation may result in weaker lower bounds compared with a direct treatment, like our sBB is capable of. The following example illustrates the disaggregation and the resulting lower bounds:

Example 6.2. Consider the function $f(x) = \log(e^x)$ over the interval $[1, e]$. By definition, this function equals $h(x) = x$ and is convex. However, by the process of disaggregation, a variable y is introduced, and f is rewritten as

$$f^*(y) = \log(y) \quad \text{and} \quad y = e^x \quad \text{with} \quad x \in [1, e].$$

The concave function $\log(y)$ is then underestimated over the interval $y \in [e, e^e]$ by its convex envelope given by the linear function

$$\frac{e - 1}{e^e - e} \cdot (y - e) + 1.$$

Finally, the convex underestimator of $f(x) = \log(e^x)$ over the interval $[1, e]$ is given by

$$\frac{e - 1}{e^e - e} \cdot (e^x - e) + 1.$$

The largest distance to the function $h(x) = x$—the convex envelope of $f(x)$—to this underestimator is approximately at $x \approx 2$ and amounts to ≈ 0.35. Consequently, in the worst case, the convex underestimator resulting from disaggregation is around 17% smaller than the convex envelope. □

In the following, we compare Gurobi's global solver (G-sBB) regarding computation time and root-node lower bounds with a static piecewise linear approximation using 10,000 equidistant segments. We solve this piecewise linear approximation with our sBB algorithm. We do not test the other PLF formulations as extensive comparisons for 10,000 segments were already provided in Tables 1, 3, and 5. As the other mentioned global solvers are also based on sBB and disaggregation, we treat Gurobi as a representative for this algorithm class and do not test the other solvers.

Therefore, consider the knapsack problem from Section 6.2.2 with the approximated functions from Table 2 again. Next to the PLF approximation of these nonlinear functions, we hand them over to a global MINLP solver that treats them directly within the algorithm. For those experiments, we construct knapsack problems as described in Section 6.2.2 but only consider functions 2, 9, 11, 12, 13, 14, 15, and 20 of Table 2 as we encountered numerical issues in Gurobi with the other functions. We believe that this is because of the relative novelty of Gurobi's solver and the difficult concatenation of elementary functions (exp, log, etc.) within the functions in Table 2. This causes problems with disaggregation.

Table 8 provides results for different numbers of variables, each for 50 random instances. Table 9 presents descriptive statistics of the lower bound obtained at the root node of our sBB and the lower bound at the root node of Gurobi's sBB method and presents the difference between them. Table 9 explains why Gurobi's sBB solver is not competitive for this knapsack problem. It can be seen that the root-node bound of Gurobi is always considerably lower than that of the sBB. Whereas our sBB computes the convex envelope of the PLF (and thus, approximately a convex envelope of the original non-PLF function), this cannot be said about Gurobi's sBB solver, which employs disaggregation. Consequently, the underestimator is less tight and results in weaker

Table 8. Solve Times (Seconds) for Knapsack Problems with Gurobi's MINLP Global Solver

Method	Med.	Avg.	Std.	Win	Fail
		Panel A: 10 variables			
G-sBB	**5.15**	147.77	397.56	28	2
sBB	**9.20**	9.47	2.05	22	0
		Panel B: 11 variables			
sBB	**10.54**	10.60	2.32	27	0
G-sBB	**17.16**	312.19	584.67	23	5
		Panel C: 12 variables			
sBB	**12.44**	13.05	3.50	34	0
G-sBB	**55.65**	450.99	698.20	16	8
		Panel D: 13 variables			
sBB	**14.07**	13.99	2.50	37	0
G-sBB	**243.29**	781.63	834.28	13	18
		Panel E: 14 variables			
sBB	**14.64**	15.32	3.74	46	0
G-sBB	**579.94**	914.11	804.96	4	20
		Panel F: 15 variables			
sBB	**16.33**	16.76	3.73	45	0
G-sBB	**1,800**	1,236	794.76	5	32
		Panel G: 20 variables			
sBB	**22.53**	23.65	7.82	49	0
G-sBB	**1,800**	1,665	441.88	1	45
		Panel H: 30 variables			
sBB	**36.15**	40.40	16.15	50	0
G-sBB	**1,800**	1,800	0.59	0	50

Notes. sBB is the proposed method in this paper. Avg., arithmetic mean; Fail, number of instances that cannot be solved by a method within the time limit; Med., median; Std., standard deviation; Win, number of instances in which each method was the fastest. The methods are sorted according to the bold numbers.

Table 9. Distribution of Root-Node Lower Bounds of Our sBB and Gurobi's sBB and the Differences Between Them

Method	Min.	Med.	Avg.	Max.	Std.
		Panel A: 10 variables			
sBB	−149	**−92**	−94	−31	32
G-sBB	−13,083	**−3,586**	−3,813	−147	3,239
Diff	39	**3,496**	3,718	13,039	3,255
		Panel B: 11 variables			
sBB	−155	**−87**	−83	5	37
G-sBB	−9,803	**−3,624**	−4,267	−84	2,672
Diff	68	**3,557**	4,184	9,744	2,679
		Panel C: 12 variables			
sBB	−160	**−90**	−91	5	40
G-sBB	−16,081	**−6,535**	−5,871	−205	3,563
Diff	61	**6,398**	5,780	16,039	3,573
		Panel D: 13 variables			
sBB	−166	**−84**	−87	8	42
G-sBB	−13,137	**−5,275**	−5,409	−198	3,374
Diff	119	**5,182**	5,322	13,060	3,388
		Panel E: 14 variables			
sBB	−175	**−117**	−104	−15	40
G-sBB	−13,184	**−4,156**	−6,260	−557	3,337
Diff	422	**4,044**	6,157	13,073	3,351
		Panel F: 15 variables			
sBB	−236	**−104**	−114	−36	47
G-sBB	−13,455	**−6,962**	−7,252	−331	3,571
Diff	250	**6,859**	7,139	13,370	3,584
		Panel G: 20 variables			
sBB	−257	**−167**	−165	44	55.37
G-sBB	−16,562	**−7,450**	−8,190	−532	3,981
Diff	347	**7,331**	8,025	16,516	3,998
		Panel H: 30 variables			
sBB	−379	**−263**	−252	−51	89
G-sBB	−28,956	**−13,225**	−13,215	−4,169	5,360
Diff	384	**12,981**	12,963	28,792	5,390

Notes. sBB is the proposed method in this paper. Avg., arithmetic mean; Diff, difference; Med., median; Std., standard deviation. The methods are sorted according to the bold numbers.

lower bounds. This, in turn, leads to longer run times as it takes longer to close the gap between upper and lower bounds.

To obtain tight lower bounds of concatenated univariate functions, like in Table 2, global MINLP solvers could either (i) approximate them by a PLF, thus obtaining an approximation of the convex envelope, or (ii) use a method like that introduced in Gounaris and Floudas (2008) to directly compute the (possibly piecewise nonlinear) convex envelope of *f*.

6.5. Discussion
Because of the difference in implementation quality—a rudimentary sBB implementation in Python compared with a commercial branch-and-cut solver in a low-level language (such as C)—it is difficult to draw firm conclusions from these computational results. Nevertheless, we sketch a summary of our observations.

Tables 1 and 3 indicate a superior scalability of the sBB; each added segment leads to a relative improvement in the computation time of the sBB compared with logarithmic approaches. This is further illustrated in *performance profiles* given in Figures 2 and 3. This superior scalability can be attributed to the sBB's slim and sparse LP relaxations, which may not always grow linearly with the number of segments (see Section 3.2). The value of a method with good scalability is illustrated in Table 4; significant improvements in solution quality are possible by refining the PLF, even if it already contains many segments. This is usually even more true for obtaining an appropriate optimality certificate.

As discussed in Section 3.2, the incremental and SOS2 models, which guarantee sharpness in the entire search tree, usually outperform logarithmic models for problems with few segments. Because the sBB also guarantees these sharpness properties, one might expect similar results for problems with smaller segments. One could even assume that this effect is enhanced because spatial branching can additionally lead to more balanced search trees by branching at the previous solution instead of at the breakpoints (see Section 5.2). However, the computational results do not support this claim. We believe that the poor performance of the sBB compared with logarithmic approaches on problems with few segments is because of the superior implementation of Gurobi's branch-and-cut solver. When the sBB is integrated into a full-featured solver, such as Gurobi or BARON, the advantage of a balanced search tree may lead the sBB to outperform logarithmic models even on problems with few segments as the SOS2 model and the incremental model do. In fact, a closer look at the performance of the sBB implementation (cf. Table 6) reveals that up to 50% of the solution time is spent on the Python-Gurobi interface. This is significant time that could be saved by integrating our sBB algorithm into a full-featured solver.

The computational results for discontinuous l.s.c PLFs show that the discontinuity results in more difficult to solve instances (cf. Table 7 versus Tables 3 and 5). None of the 50 instances for 5,000 segments could be solved by Gurobi within the 1,800 seconds; it is the same for the 50 instances of 10,000 segments. The relative performance of our sBB to Gurobi's PLF solver is similar to the continuous PFL instances in that our sBB is superior for 500 and more segments (cf. Tables 1, 3, and 5).

The comparisons with Gurobi as a global solver confirmed the good scalability of our sBB method (Table 8). Although the running time of our sBB method scales approximately linearly with the number of variables, the global solver scales approximately exponentially. Already with 11 variables, our sBB is clearly superior. Remarkable is the extremely low standard deviation of the running times of our sBB, which shows that the computational performance is very consistent among the 50 instances tested. The superior performance of our sBB can be explained by the better lower bounding (cf. Table 9).

7. Approximating Separable Functions

We mentioned earlier in Section 1.3 the need for computationally efficient scalable algorithms and the various error bounds that have been calculated in the literature to determine the number of breakpoints needed from a good PLF approximation. We present an error bound for the number of breakpoints required in a PLF approximation to achieve a desired error to the problem of optimizing a separable function. Our bound is different than existing results because we do not assume differentiability of the function that is being approximated. Instead, we work with Hölder continuous functions, which are defined as follows.

Definition 7.1. A function $h : X \to \mathbb{R}$ over a closed set $X \subseteq \mathbb{R}^n$ is said to be (α, β)-Hölder continuous for some constants $\alpha, \beta > 0$ if

$$|f(x) - f(x')| \leqslant \beta \|x - x'\|_2^\alpha \qquad x, x' \in X.$$

The function is Lipschitz continuous when $\alpha = 1$, whereas for $\alpha > 1$, the function must be constant over its domain. We assume $\alpha \in (0, 1]$.

For some closed convex set S and hyperrectangle H, consider the nonconvex separable minimization problem

$$\phi^* := \min \phi(x) := \sum_{i=1}^{n} \phi_i(x_i) \quad \text{s.t.} \quad x \in S \cap H,$$

where for each $i = 1, \ldots, n$, the univariate function $\phi_i : [l_i, u_i] \to \mathbb{R}$, whose domain is some closed interval $[l_i, u_i] \subset \mathbb{R}$, is (α_i, β_i)-Hölder continuous. This means that

$$|\phi_i(t) - \phi_i(t')| \leqslant \beta_i |t - t'|^{\alpha_i}, \qquad t, t' \in [l_i, u_i].$$

Let x^* denote its optimal solution, which exists because ϕ is continuous and $S \cap H$ is compact. Suppose that for each ϕ_i, we construct a continuous PLF approximation $\widehat{\phi}_i : [l_i, u_i] \to \mathbb{R}$ with $K_i + 1$ breakpoints that are indexed by the set $\{b_i^k : k = 0, 1, \ldots, K_i\}$. This PLF is constructed in the natural way by joining consecutive breakpoints so that the kth segment is obtained by joining the points $(b_i^{k-1}, \phi_i(b_i^{k-1}))$ and $(b_i^k, \phi_i(b_i^k))$ for $k = 1, \ldots, K_i$. Summing these over $i = 1, \ldots, n$ creates the PLF $\widehat{\phi}(x) = \sum_{i=1}^{n} \widehat{\phi}_i(x_i)$, whose optimization yields a finite value $\widehat{\phi}$ and solution $\widehat{x}$:

$$\widehat{\phi} = \widehat{\phi}(\widehat{x}) := \min \sum_{i=1}^{n} \widehat{\phi}_i(x_i) \quad \text{s.t.} \quad x \in S \cap H.$$

There is no immediate relation between ϕ^* and $\widehat{\phi}$, but we can deduce two inequalities. First, the optimal solution $\widehat{x}$ of the PLF problem being feasible to $S \cap H$ implies that the optimum of the original problem can be upper bounded.

Observation 7.1. $\phi^* \leqslant \phi(\widehat{x})$.

Second, if the approximate solution $\widehat{x}$ belongs to subintervals of concavity,[1] then we can also lower bound the global optimum.

Observation 7.2. $\widehat{\phi} \leqslant \phi^*$ if for each $i = 1, \ldots, n$, ϕ_i is concave over the subinterval $[b_i^k, b_i^{k+1}]$ containing $\widehat{x}_i$.

Proof. This is because the stated assumption implies $\widehat{\phi}(x^*) \leqslant \phi(x^*)$, and we know from the optimality of $\widehat{x}$ that $\widehat{\phi}(\widehat{x}) \leqslant \widehat{\phi}(x^*)$. □

In general, $\widehat{\phi} := \widehat{\phi}(\widehat{x})$ is neither a lower bound nor an upper bound on ϕ^*. Our main result here is that to control the additive gap on $\widehat{\phi}$, there is a formula for the number of breakpoints in the PLFs that depends on the continuity parameters and the width of interval bounds.

Proposition 7.1. *Let $\varepsilon, \delta > 0$ be given, and denote*

$$\theta_i := \frac{u_i - l_i}{\delta}, \qquad \rho_i := \sqrt[\alpha_i]{\frac{\beta_i}{\varepsilon}[1 + \theta_i^{1-\alpha_i}]} \quad i = 1, \ldots, n.$$

Solving the PLF approximate problem by creating for each i at least $\theta_i - \frac{n^{-1/\alpha_i}}{\delta \rho_i}$ segments such that the breakpoints are spaced at least δ-apart yields an approximate value $\widehat{\phi}$ that satisfies $\widehat{\phi} \geqslant \phi^* - \varepsilon$.

We argue this by establishing the approximation error for univariate functions and then, gluing together the individual pieces.

7.1. PLF Approximations of Univariate Functions

Suppose that we are given a univariate function $f : I \to \mathbb{R}$ that is (α, β)-Hölder continuous on the interval $I := [l, u] \subset \mathbb{R}$, meaning that $|f(t) - f(t')| \leqslant \beta |t - t'|^\alpha$ for all $t, t' \in I$. For any finite integer $K \geqslant 1$ and $\delta > 0$, let

$$\mathcal{B}_{K, \delta} := \{B := \{b^0, b^1, \ldots, b^K\} : b^0 = l, b^K = u, b^{i+1} - b^i \geqslant \delta \ \forall i\}$$

be the collection of all sets of $K + 1$ breakpoints (sorted in increasing order) in interval I that are at least δ apart from each other. For every $B \in \mathcal{B}_{K, \delta}$, we have a continuous PLF $g_B : I \to \mathbb{R}$ that approximates f by interpolation with $K + 1$ breakpoints. In particular, the K segments of g_B are obtained by joining consecutive points so that for $i = 1, \ldots, K$, the ith segment joins the points $(b^{i-1}, f(b^{i-1}))$ and $(b^i, f(b^i))$ with a line segment whose slope is $m_i := (f(b^i) - f(b^{i-1}))/(b^i - b^{i-1})$. Each of these slopes can be upper bounded by parameters for f, which leads to a Lipschitz constant for g_B that is independent of K.

Lemma 7.1. *For every $B \in \mathcal{B}_{K, \delta}$, g_B has a Lipschitz constant equal to $\beta/\delta^{1-\alpha}$.*

Proof. Let us begin with the following general result, which may be known, but because we could not find a reference, a self-contained proof is given in the appendix for completeness. □

Claim 7.1 (Lipschitz Continuity of PLF). A continuous univariate PLF on a closed interval has its smallest Lipschitz constant equal to the maximum absolute value of the slope of its linear segments.

We derive another technicality.

Claim 7.2. $|m_i| \leqslant \beta/\delta^{1-\alpha}$ for all $i = 1, \ldots, K$.

Proof of Claim 7.2. By construction of g_B, we have $g_B(b^i) = f(b^i)$ and $g_B(b^{i+1}) = f(b^{i+1})$, and so, the definition of slope gives us $|f(b^{i+1}) - f(b^i)| = |m_i|(b^{i+1} - b^i)$. The Hölder property leads to $|m_i|(b^{i+1} - b^i) \leqslant \beta (b^{i+1} - b^i)^\alpha$, which after noting $\alpha \in (0, 1]$, reduces to

$$|m_i| \leqslant \beta (b^{i+1} - b^i)^{\alpha-1} = \frac{\beta}{(b^{i+1} - b^i)^{1-\alpha}} \leqslant \frac{\beta}{\delta^{1-\alpha}},$$

where the last inequality uses $B \in \mathcal{B}_{K, \delta}$. □

Our assertion follows after combining the above two claims.

We will need one more technical result.

Lemma 7.2. *Let $X \subset \mathbb{R}^n$ be a compact set and R be the radius of a ball with its center in X such that the ball encloses X. Let $h_1 : X \to \mathbb{R}$ be L-Lipschitz over X and $h_2 : X \to \mathbb{R}$ be (α, β)-Hölder continuous over X. Then, $h_1 - h_2 : x \in X \longmapsto h_1(x) - h_2(x)$ is $(\alpha, L(2R)^{1-\alpha} + \beta)$-Hölder continuous over X.*

Proof. For any $x, x' \in X$, we have

$$|(h_1 - h_2)(x) - (h_1 - h_2)(x')| = |h_1(x) - h_1(x') - (h_2(x) - h_2(x'))|$$

$$\leqslant |h_1(x) - h_1(x')| + |h_2(x) - h_2(x')|$$

$$\leqslant L\|x - x'\| + \beta\|x - x'\|^\alpha,$$

where the first inequality is the triangle inequality for absolute values and the second inequality is from Lipschitz and Hölder continuity of h_1 and h_2. The distance between any $x, x' \in X$ can be bounded as $\|x - x'\| \leqslant 2R$ using the triangle inequality. Therefore, for any $\alpha \in (0, 1]$,

$$\left(\frac{\|x - x'\|}{2R}\right)^\alpha \geqslant \frac{\|x - x'\|}{2R} \Longrightarrow \|x - x'\| \leqslant (2R)^{1-\alpha} \|x - x'\|^\alpha.$$

Substituting this into the above inequality gives us

$$|(h_1 - h_2)(x) - (h_1 - h_2)(x')| \leqslant L(2R)^{1-\alpha} \|x - x'\|^\alpha + \beta\|x - x'\|^\alpha = (L(2R)^{1-\alpha} + \beta)\|x - x'\|^\alpha,$$

and hence, our claim is that $h_1 - h_2$ is Hölder continuous with parameters α and $L(2R)^{1-\alpha} + \beta$. $\square$

Now, let us derive our error bound for a univariate function. The error of a continuous PLF with respect to f is defined as the largest additive approximation gap over the domain. Thus, we have the error function $\xi : \mathbb{N} \times \mathbb{R}_{>0} \to \mathbb{R}_{\geqslant 0}$ given by

$$\xi : (K, \delta) \in \mathbb{N} \times \mathbb{R}_{>0} \longmapsto \max_{B \in \mathcal{B}_{K,\delta}} \max_{x \in I} |f(x) - g_B(x)|.$$

To state our lower bound for the number of breakpoints required to achieve a given error, let us introduce two parameters dependent on the minimum spacing parameter δ:

$$\theta = \theta(\delta) := \frac{u - l}{\delta}, \qquad \rho = \rho(\delta) := \sqrt[\alpha]{\frac{\beta}{\varepsilon}[1 + \theta^{1-\alpha}]}.$$

Proposition 7.2. *Given any $\varepsilon, \delta > 0$, we have $\xi(K, \delta) \leqslant \varepsilon$ if*

$$K > \theta - \frac{1}{\delta \rho}.$$

Proof. Because g_B is a PLF that can be described as $g_B(x) = m_i x + f(b^i) - m_i b^i$ when $x \in [b^{i-1}, b^i]$ for any i, the error function can be written as

$$\xi(K, \delta) = \max_{B \in \mathcal{B}_{K,\delta}} \max_{i=1,\dots,K} \max_{x \in [b^{i-1}, b^i]} |f(x) - [m_i x + f(b^i) - m_i b^i]|.$$

Consider the function $h_i : x \in [b_i, b_{i+1}] \longmapsto f(x) - m_i x - f(b^i) + m_i b_i$ that appears in the error function. This is the difference of an (α, β)-Hölder continuous function and a linear function, which is Lipschitz continuous with constant $|m_i|$. Applying Lemma 7.2 with $R = (u - l)/2$ for the interval I, we obtain h_i to be Hölder continuous with parameters α and $|m_i|(u - l)^{1-\alpha} + \beta$. Using the definition of Hölder continuity for any $x \in [b_i, b_{i+1}]$ leads to

$$|h_i(x)| = |h_i(x) - h_i(b_i)| \leqslant (|m_i|(u - l)^{1-\alpha} + \beta)(x - b_i)^\alpha \leqslant (|m_i|(u - l)^{1-\alpha} + \beta)\Delta(B)^\alpha,$$

where for the first equality, we have used $h_i(b_i) = 0$ because of exactness of PLF at breakpoints, and in the last inequality, we denote $\Delta(B) := \max_{i=1,\dots,K} b^i - b^{i-1}$ to be the maximum distance between consecutive breakpoints. We have $u - l = \sum_{i=1}^K b^i - b^{i-1} \geqslant \Delta(B) + (K - 1)\delta$ because of $B \in \mathcal{B}_{K,\delta}$. This implies that $\Delta(B) \leqslant u - l - (K - 1)\delta$. Substituting this upper bound into the above leads to $|h_i(x)| \leqslant (|m_i|(u - l)^{1-\alpha} + \beta)(u - l - (K - 1)\delta)^\alpha$. Because $\xi(K, \delta) = \max_B \max_i \max_x |h_i(x)|$, after using Claim 7.2, which gives an upper bound on $|m_i|$ that is independent of i, it follows that

$$\beta(\theta^{1-\alpha} + 1)(u - l - (K - 1)\delta)^\alpha \leqslant \varepsilon$$

is a sufficient condition for $\xi(K, \delta) \leqslant \varepsilon$. Rearranging terms yields our lower bound on K. $\square$

When uniformly spaced breakpoints are to be considered only, the above proof can be modified at the step where we upper bound the maximum separation $\Delta(B)$. In particular, we have $\Delta(B) = (u - l)/K$ in the uniform case, and the remaining proof carries through. Hence, we can bound as follows the error $\tilde{\xi}(K) := \max_{x \in I} |f(x) - g_B(x)|$, where B is the unique set of K breakpoints that are uniformly spaced (note that δ is not needed as an input parameter in the uniform case).

Corollary 7.1. $(u - l)\rho$ *uniformly spaced segments guarantee an additive error of at most* ε.

7.2. Proof of Proposition 7.1

We have $\phi^* \leqslant \phi(\widehat{x})$ from Observation 7.1. For every i, we can apply Proposition 7.2 to control the approximation error to ε/n by selecting the number of segments K_i to be large enough. Our claim follows after recognizing that the errors are additive and ϕ is a separable function.

8. Conclusion and Future Work

In this paper, a new perspective on piecewise linear optimization is taken. We adopt a global and nonlinear continuous approach instead of discrete optimization. The developed spatial branch-and-bound algorithm has small, sparse, and sharp LP relaxations throughout the search tree. Computational experiments have shown that even a rudimentary sBB implementation in Python can outperform state-of-the-art logarithmic models solved by Gurobi if the number of segments is sufficiently high. Nonetheless, we advocate a problem-specific approach when selecting a solution method for separable piecewise linear optimization problems. If the PLFs involved have many segments, the sBB could be the method of choice because of its slim and sparse LP relaxations. However, for PLFs with few segments, MILP models, such as the classical incremental model, might be faster because of their large formulation and thus, the better possibilities for cutting planes.

Discrete approaches in piecewise linear optimization have witnessed over 60 years of fruitful research, which led to the current state of the art. In contrast, this paper is an initial attempt toward an efficient method that is based on continuous optimization techniques and is globally convergent. We recognize that our implementation is rudimentary at this stage and can benefit from several enhancements and sophistications that would accelerate its performance. Therefore, there are still some open questions. Further research can focus on extensions to nonseparable cases, cutting planes, specialized branching rules, integration in a full branch-and-cut solver, or further development of sBB algorithms for discontinuous functions. We leave these for future research but outline some of these ideas in the next paragraphs.

The ideas of pseudocost, strong, and reliability branching from MILP (Achterberg et al. 2005) could be adopted here. Moreover, there have been many works (Benson 1990, Kesavan et al. 2004, D'Ambrosio et al. 2020) on strengthening the relaxations for separable nonconvex terms in a branch-and-cut algorithm, and it is conceivable that some of these ideas can be applied to separable PLFs to accelerate our sBB. This would be a counterpart to the valid inequalities and cutting planes that have been developed for MILP and SOS2 models.

Future work could also extend our work to non-l.s.c. PLFs. Although our sBB can generate polyhedral relaxations of any separable PLF, we currently do not have a branching rule that gives asymptotic convergence when the PLF is non-l.s.c. This does not seem to be an easy task because convergence issues for relaxations of discontinuous functions are well known and also, easy to see with simple examples (cf. Figure 1). Nonetheless, it may be worth tackling this problem at least for separable PLFs because the SOS2 branching rule has been generalized (de Farias et al. 2008), although only as a proof of concept and not something that has been implemented in MILP solvers. Moreover, one could explore machine learning techniques for branching decisions as was done recently for nonconvex polynomial optimization problems (Ghaddar et al. 2023).

Lastly, our approach could be extended to handle nonseparable PLFs. Although the Graham's scan algorithm is limited to two dimensions and thus, only applicable to univariate or separable PLFs, other algorithms, such as Quickhull (Barber et al. 1996), can compute the convex hull in multiple dimensions. This makes them suitable for identifying the convex envelope of nonseparable PLFs.

Appendix. Proof of Claim 7.1

Claim 7.1 (Lipschitz Continuity of PLF). *A continuous univariate PLF on a closed interval has its smallest Lipschitz constant equal to the maximum absolute value of the slope of its linear segments.*

Proof. Let h be a continuous PLF on $I := [l, u]$ formed by breakpoints $\{b^0, b^1, \ldots, b^K\}$, where $b^i < b^{i+1}$, $b^0 = l$, and $b^K = u$. Denote the slope of the ith segment by $m_i := \frac{h(b^i) - h(b^{i-1})}{b^i - b^{i-1}}$. Take any distinct $x, x' \in I$ with $x' \in [b^{k-1}, b^k]$ and $x \in [b^{j-1}, b^j]$ for

some $1 \leqslant k \leqslant j \leqslant K$. The case $k = j$ is trivial because of linearity in each piece, so assume $k < j$. We have

$$h(x) - h(x') = [h(x) - h(b^{j-1})] + [h(b^{j-1}) - h(b^{j-2})] + \cdots + [h(b^k) - h(x')]$$

$$= m_j(x - b^{j-1}) + m_{j-1}(b^{j-1} - b^{j-2}) + \cdots + m_k(b^k - x')$$

$$\leqslant \left[\max_{i=k,\ldots,j} m_i \right] (x - b^{j-1} + b^{j-1} - b^{j-2} + \cdots + b^k - x')$$

$$= \left[\max_{i=k,\ldots,j} m_i \right] (x - x').$$

Switching the roles of x and x' and following similar steps give us

$$h(x') - h(x) \leqslant \left[\max_{i=k,\ldots,j} -m_i \right] (x - x').$$

Recall that any four reals (a_1, a_2, a_3, a_4) with $a_1 \leqslant a_2$ and $a_3 \leqslant a_4$ also satisfy $\max\{a_1, a_3\} \leqslant \max\{a_2, a_4\}$. Using this fact with the above two inequalities gives us

$$|h(x) - h(x')| \leqslant \max\left\{ \max_{i=k,\ldots,j} m_i, \; \max_{i=k,\ldots,j} (-m_i) \right\} (x - x')$$

$$= \left[\max_{i=k,\ldots,j} |m_i| \right] (x - x')$$

$$\leqslant \left[\max_{i=1,\ldots,K} |m_i| \right] (x - x').$$

Because x and x' are arbitrary in I, the correctness of the Lipschitz constant follows from above. This is also the best-possible constant because we can take x and x' to be between the breakpoints where the slope has the highest absolute value. $\quad\square$

Endnote

[1] Every continuous univariate function on an interval can be partitioned into subintervals such that over each subinterval, it is either convex or concave.

References

Achterberg T, Koch T, Martin A (2005) Branching rules revisited. *Oper. Res. Lett.* 33(1):42–54.

Adams W, Gupte A, Xu Y (2019) Error bounds for monomial convexification in polynomial optimization. *Math. Programming* 175(1–2):355–393.

Adjiman CS, Dallwig S, Floudas CA, Neumaier A (1998) A global optimization method, αBB, for general twice-differentiable constrained NLPs—I. Theoretical advances. *Comput. Chemical Engrg.* 22(9):1137–1158.

Al-Khayyal FA, Sherali HD (2000) On finitely terminating branch-and-bound algorithms for some global optimization problems. *SIAM J. Optim.* 10(4):1049–1057.

Barber CB, Dobkin DP, Huhdanpaa H (1996) The quickhull algorithm for convex hulls. *ACM Trans. Math. Software* 22(4):469–483.

Bärmann A, Burlacu R, Hager L, Kleinert T (2023) On piecewise linear approximations of bilinear terms: Structural comparison of univariate and bivariate mixed-integer programming formulations. *J. Global Optim.* 85(4):789–819.

Beach B, Hildebrand R, Huchette J (2022) Compact mixed-integer programming formulations in quadratic optimization. *J. Global Optim.* 84(4):869–912.

Beale E, Forrest JJ (1976) Global optimization using special ordered sets. *Math. Programming* 10(1):52–69.

Benson HP (1990) Separable concave minimization via partial outer approximation and branch and bound. *Oper. Res. Lett.* 9(6):389–394.

Burer S, Letchford AN (2012) Non-convex mixed-integer nonlinear programming: A survey. *Surveys Oper. Res. Management Sci.* 17(2):97–106.

Burlacu R, Geißler B, Schewe L (2020) Solving mixed-integer nonlinear programmes using adaptively refined mixed-integer linear programmes. *Optim. Methods Software* 35(1):37–64.

Casado LG, Martínez JA, García I, Sergeyev YD (2003) New interval analysis support functions using gradient information in a global minimization algorithm. *J. Global Optim.* 25(4):345–362.

Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) *Introduction to Algorithms* (MIT Press, Cambridge, MA).

Croxton KL, Gendron B, Magnanti TL (2003) A comparison of mixed-integer programming models for nonconvex piecewise linear cost minimization problems. *Management Sci.* 49(9):1268–1273.

Croxton KL, Gendron B, Magnanti TL (2007) Variable disaggregation in network flow problems with piecewise linear costs. *Oper. Res.* 55(1):146–157.

D'Ambrosio C, Lee J, Skipper D, Thomopulos D (2020) Handling separable non-convexities using disjunctive cuts. Baiou, M Gendron B, Gunluk O, Mahjoub A, eds. *Combinatorial Optimization: ISCO 2020*, Lecture Notes in Computer Science, vol. 12176 (Springer, Cham, Switzerland), 102–114.

Dantzig GB (1960) On the significance of solving linear programming problems with some integer variables. *Econometrica* 28(1):30–44.

de Farias IR Jr, Zhao M, Zhao H (2008) A special ordered set approach for optimizing a discontinuous separable piecewise linear function. *Oper. Res. Lett.* 36(2):234–238.

de Farias IR, Kozyreff E, Gupta R, Zhao M (2013) Branch-and-cut for separable piecewise linear optimization and intersection with semi-continuous constraints. *Math. Programming Comput.* 5(1):75–112.

Dey SS, Gupte A (2015) Analysis of MILP techniques for the pooling problem. *Oper. Res.* 63(2):412–427.

Duguet A, Ngueveu SU (2022) Piecewise linearization of bivariate nonlinear functions: Minimizing the number of pieces under a bounded approximation error. Ljubíc I, Barahona F, Dey SS, Mahjoub AR, eds. *Combinatorial Optimization: ISCO 2022*, Lecture Notes in Computer Science, vol. 13526 (Springer, Cham, Switzerland), 117–129.

Falk JE, Soland RM (1969) An algorithm for separable nonconvex programming problems. *Management Sci.* 15(9):550–569.

Feijoo B, Meyer R (1988) Piecewise-linear approximation methods for nonseparable convex optimization. *Management Sci.* 34(3):411–419.

Fourer R (1985) A simplex algorithm for piecewise-linear programming I: Derivation and proof. *Math. Programming* 33(2):204–233.

Frenzen CL, Sasao T, Butler JT (2010) On the number of segments needed in a piecewise linear approximation. *J. Comput. Appl. Math.* 234(2):437–446.

Füllner C, Rebennack S (2022) Non-convex nested Benders decomposition. *Math. Programming* 196(1):987–1024.

Geisler B, Martin A, Morsi A, Schewe L (2012) Using piecewise linear functions for solving MINLPs. Lee J, Leyffer S, eds. *Mixed Integer Nonlinear Programming*, IMA Volumes in Mathematics and Its Applications, vol. 154 (Springer, Cham, Switzerland), 287–314.

Ghaddar B, Gómez-Casares I, González-Díaz J, González-Rodríguez B, Pateiro-López B, Rodríguez-Ballesteros S (2023) Learning for spatial branching: An algorithm selection approach. *INFORMS J. Comput.* 35(5):1024–1043.

Gorissen BL (2022) Interior point methods can exploit structure of convex piecewise linear functions with application in radiation therapy. *SIAM J. Optim.* 32(1):256–275.

Gounaris CE, Floudas CA (2008) Tight convex underestimators for C^2-continuous functions: I. Univariate functions. *J. Global Optim.* 42(1):51–67.

Graham RL (1972) An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Processing Lett.* 1(4):132–133.

Grimstad B, Knudsen BR (2020) Mathematical programming formulations for piecewise polynomial functions. *J. Global Optim.* 77(3):455–486.

Gupte A, Koster AM, Kuhnke S (2022) An adaptive refinement algorithm for discretizations of nonconvex QCQP. Schulz C, Ucar B, eds. *20th International Symposium on Experimental Algorithms: SEA 2022*, Leibniz International Proceedings in Informatics (LIPIcs), vol. 233 (Schloss Dagstuhl Publishing, Wadern, Germany), 24:1–24:14.

Gurobi Optimization (2024) Documentation—General constraints. https://www.gurobi.com/documentation/current/refman/general_constraints.html.

Horst R (1986) A general class of branch-and-bound methods in global optimization with some new approaches for concave minimization. *J. Optim. Theory Appl.* 51(2):271–291.

Hübner T, Gupte A, Rebennack S (2025) Spatial branch and bound for nonconvex separable piecewise linear optimization. https://github.com/INFORMSJoC/2024.0755.

Huchette J, Vielma JP (2023) Nonconvex piecewise linear functions: Advanced formulations and simple modeling tools. *Oper. Res.* 71(5):1835–1856.

Keha AB, de Farias IR, Nemhauser GL (2004) Models for representing piecewise linear cost functions. *Oper. Res. Lett.* 32(1):44–48.

Keha AB, de Farias IR, Nemhauser GL (2006) A branch-and-cut algorithm without binary variables for nonconvex piecewise linear optimization. *Oper. Res.* 54(5):847–858.

Kesavan P, Allgor RJ, Gatzke EP, Barton PI (2004) Outer approximation algorithms for separable nonconvex mixed-integer nonlinear programs. *Math. Programming* 100(3):517–535.

Kim J, Richard J-PP, Tawarmalani M (2024) Piecewise polyhedral relaxations of multilinear optimization. *SIAM J. Optim.* 34(4):3167–3193.

Kong L, Maravelias CT (2020) On the derivation of continuous piecewise linear approximating functions. *INFORMS J. Comput.* 32(3):531–546.

Kontogiorgis S (2000) Practical piecewise-linear approximation for monotropic optimization. *INFORMS J. Comput.* 12(4):324–340.

Leyffer S, Sartenaer A, Wanufelle E (2008) Branch-and-refine for mixed-integer nonconvex global optimization. Working Paper No. ANL/MCS-P1547-0908, Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL.

Locatelli M, Schoen F (2013) *Global Optimization: Theory, Algorithms, and Applications*, MOS-SIAM Series on Optimization, vol. MO15 (SIAM, Philadephia).

Lyu B, Hicks IV, Huchette J (2025) Building formulations for piecewise linear relaxations of nonlinear functions. *Oper. Res.*, ePub ahead of print April 24, https://doi.org/10.1287/opre.2023.0187.

Magnanti TL, Stratila D (2004) Separable concave optimization approximately equals piecewise linear optimization. Bienstock D, Nemhauser G, eds. *Integer Programming and Combinatorial Optimization: IPCO 2004*, Lecture Notes in Computer Science, vol. 3064 (Springer, Berlin), 234–243.

Markowitz HM, Manne AS (1957) On the solution of discrete programming problems. *Econometrica* 25(1):84–110.

Meyer RR (1976) Mixed integer minimization models for piecewise-linear functions of a single variable. *Discrete Math.* 16(2):163–171.

Nagarajan H, Lu M, Wang S, Bent R, Sundar K (2019) An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs. *J. Global Optim.* 74(4):639–675.

Natali JM, Pinto JM (2009) Piecewise polynomial interpolations and approximations of onedimensional functions through mixed integer linear programming. *Optim. Methods Software* 24(4–5):783–803.

Ngueveu SU (2019) Piecewise linear bounding of univariate nonlinear functions and resulting mixed integer linear programming-based solution methods. *Eur. J. Oper. Res.* 275(3):1058–1071.

Posypkin M, Usov A, Khamisov O (2020) Piecewise linear bounding functions in univariate global optimization. *Soft Comput.* 24(23):17631–17647.

Rebennack S (2016) Computing tight bounds via piecewise linear functions through the example of circle cutting problems. *Math. Methods Oper. Res.* 84(1):3–57.

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

Rebennack S, Krasko V (2020) Piecewise linear function fitting via mixed-integer linear programming. *INFORMS J. Comput.* 32(2):507–530.

Rebennack S, Nahapetyan A, Pardalos PM (2009) Bilinear modeling solution approach for fixed charge network flow problems. *Optim. Lett.* 3(3):347–355.

Shectman JP, Sahinidis NV (1998) A finite algorithm for global minimization of separable concave programs. *J. Global Optim.* 12(1):1–36.

Sherali HD (2001) On mixed-integer zero-one representations for separable lower-semicontinuous piecewise-linear functions. *Oper. Res. Lett.* 28(4):155–160.

Soland RM (1971) An algorithm for separable nonconvex programming problems II: Nonconvex constraints. *Management Sci.* 17(11):759–773.

Sundar K, Sanjeevi S, Nagarajan H (2022) Sequence of polyhedral relaxations for nonlinear univariate functions. *Optim. Engrg.* 23(2):877–894.

Tawarmalani M, Sahinidis NV (2004) Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Math. Programming* 99(3):563–591.

Thakur LS (1978) Error analysis for convex separable programs: The piecewise linear approximation and the bounds on the optimal objective value. *SIAM J. Appl. Math.* 34(4):704–714.

Toriello A, Vielma JP (2012) Fitting piecewise linear continuous functions. *Eur. J. Oper. Res.* 219(1):86–95.

Tuy H (2016) *Convex Analysis and Global Optimization*, 2nd ed., Springer Optimization and Its Applications, vol. 110 (Springer, Cham, Switzerland).

Tuy H, Horst R (1988) Convergence and restart in branch-and-bound algorithms for global optimization. Application to concave minimization and D.C. ptimization problems. *Math. Programming* 41(1):161–183.

Vielma JP, Nemhauser GL (2011) Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. *Math. Programming* 128(1–2):49–72.

Vielma JP, Ahmed S, Nemhauser G (2010) Mixed-integer models for nonseparable piecewise-linear optimization: Unifying framework and extensions. *Oper. Res.* 58(2):303–315.

Vielma JP, Keha AB, Nemhauser GL (2008) Nonconvex, lower semicontinuous piecewise linear optimization. *Discrete Optim.* 5(2):467–488.

Warwicker JA, Rebennack S (2022) A comparison of two mixed-integer linear programs for piecewise linear function fitting. *INFORMS J. Comput.* 34(2):1042–1047.

Warwicker JA, Rebennack S (2024) Efficient continuous piecewise linear regression for linearising univariate non-linear functions. *IISE Trans.* 57(3):231–245.

Wechsung A, Barton PI (2014) Global optimization of bounded factorable functions with discontinuities. *J. Global Optim.* 58(1):1–30.

Yıldız S, Vielma JP (2013) Incremental and encoding formulations for mixed integer programming. *Oper. Res. Lett.* 41(6):654–658.

Zhao M, de Farias IR Jr (2013) The piecewise linear optimization polytope: New inequalities and intersection with semi-continuous constraints. *Math. Programming* 141(1–2):217–255.