Spatial branch-and-bound for nonconvex separable piecewise linear optimization

Thomas Hübner* Akshay Gupte† Steffen Rebennack‡

May 1, 2024

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 optimisation, 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 (sBB) 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 semi-continuous. A Python implementation of our algorithm is tested on knapsack and network flow problems and compared with logarithmic MILP formulations solved by a commercial MILP solver. The numerical experiments indicate significant performance gains up to an order of magnitude for medium- to large-sized PLFs.

Keywords. Piecewise linear functions, Spatial branch-and-bound, Global optimization, Lower semi-continuity, Convex underestimators, Branching rules

AMS 2020 subject classification. 90C26, 90C57

1 Introduction

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 approximation of a nonlinear nonconvex function [Gei+12, DG15, Nag+19, BGS20, BHH22, Bär+23, WR24]. When a PLF is convex and is either minimized or appears in a $\leqslant$ constraint, it can be modelled as a linear program (LP). In general, a PLF is NP-hard to optimize [KFN06] 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 [MM57, Dan60, BF76]. Univariate PLFs also arise as approximations to one-dimensional nonconvex functions in a global optimization problem [LSW08, NP09, RK15, GK20, PUK20, SSN22]. In fact, a separable concave

*Power Systems Laboratory, ETH Zürich, Switzerland, thuebner@ethz.ch

†School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh, United Kingdom, akshay.gupte@ed.ac.uk

‡Institute for Operations Research, Karlsruhe Institute of Technology, Germany, steffen.rebennack@kit.edu

function minimization can be approximated to an arbitrary precision by a single separable PLF problem [MS04].

The common way to approach problems with nonconvex PLFs is by developing exact formulations based on either *mixed-integer linear programming* (MILP) models 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 a MILP solver. Classical MILP and SOS2 modelling approaches (see surveys in [VAN10, Reb16]) initially focused on continuous separable PLFs, but were later extended by [VAN10] to the non-separable case. It is known that their LPs provide the same relaxation strength [She01, CGM03, KFN04]. However, they have the drawback of using as many binary variables as the number of segments of a PLF. This was remedied by [VN09, HV22] 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 [KFN06, VKN08, ZF12, Far+13]. Extensions of MILP models to lower semicontinuous (l.s.c.) PLFs have been studied [VKN08, VAN10]. For general discontinuous PLFs, one cannot expect a MILP formulation with bounded integer variables [Mey76, Theorem 2.1], but the SOS2 branching scheme has been adapted [FZZ08]. Many of these modelling and algorithmic advances have been implemented in state-of-the-art MILP solvers, and leveraged to build stronger polyhedral relaxations of nonconvex functions [Reb16, KRT22, LHH23].

1.1 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 a MILP or SOS 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 generate convex underestimators for it and refine them to develop a *spatial branch-and-bound* (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, (iii) more freedom in branching decisions. Through extensive computational experiments we demonstrate that even a rudimentary Python implementation of the sBB can provide speed-ups of an order of magnitude over modern logarithmic models solved by Gurobi if the number of segments is sufficiently high and that these speed-ups grow with every segment added to the PLFs.

The existing approaches for PLF optimization would 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 semi-continuous 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 carries over to our PLF optimization problem also. 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. § 2.2 introduces the basic concepts of our sBB algorithm and relates it to the MILP and SOS2 approaches. § 3 studies various properties of a univariate PLF that underpin our algorithm. The sBB algorithm, with all its elements, is described and analyzed for convergence in § 4. Computational testing is done in § 5 where comparisons are also drawn with logarithmic-sized MILP models. Lastly, conclusions and some future directions are mentioned in § 6.

1.2 Need for scalable algorithms

Since 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). The latter can be achieved only when the nonconvexities are in the objective only and not in the constraints. Large 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 since that is the focus of this paper, but note that some error-bounding analysis has also been done for higher-dimensional functions [DG15, AGX19, Bär+23].

The errors in the PLF relaxation of a univariate function is 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. [FSB10, 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. For functions that are Lipschitz continuous, which is a weaker condition than differentiability, Proposition 3.9 derives a non-asymptotic formula for the number of breakpoints in a PLF approximation with uniformly spaced breakpoints. This shows that c/ε many breakpoints is sufficient, for $c = \lceil 2Lw \rceil$ with L being the Lipschitz constant and w the width of the interval.

There are computationally intensive MILP-based methods for computing best-fit PLFs [TV12, KM20, RK20, WR22]. 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. Recognising this obstacle, some recent studies [Nag+19, BGS20, GKK22] 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 Background & Overview

We consider the separable nonconvex piecewise linear optimization problem given by

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

where every $f_i : [l_i, u_i] \to \mathbb{R}$ is a univariate PLF, possibly nonconvex and discontinuous, over an interval $H_i := [l_i, u_i]$. The feasible set is the intersection of a closed convex set $S \subset \mathbb{R}^n$ and a hyper-rectangle $H := \{x \in \mathbb{R}^n : \ l_i \leqslant x_i \leqslant u_i, \ i = 1, \ldots, n\}$. 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 pre-processing 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 $\{y_i^k : \ k \in \mathcal{K}_i\}$. Since we allow discontinuities at the breakpoints, we also need to know the left and right limits at each breakpoint to characterise f_i. The left limit is denoted by $y_i^{k,-}$ and the right limit by $y_i^{k,+}$. For the left (resp. right) endpoint, we set the left (resp. right) limit to the function value. Thus, for every $k \in \mathcal{K}_i$ we have as input the tuple

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

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

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

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 limit coincide with the function value.

2.1 Background on sBB

The use of a sBB for optimizing a separable function (not necessarily PLF) was first done by Falk and Soland [FS69]. This was improved upon by [Hor86, TH88] to general nonconvex functions and since then sBB algorithms for global optimization have matured immensely [cf. LS13, Tuy16] and there are many sophisticated implementations in global solvers for optimizing smooth functions. The sBB is similar to the integer branch-and-bound (B&B) but has some major differences. 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. Secondly, branching takes place on continuous variables (hence the term *spatial*) which leads to a partition of the feasible region in hyperrectangles. Thirdly, 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, since 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 [LS13, chap. 5] 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 hyper-rectangles used for branching converges to a point), and
2. *exactness in the limit* for the underestimators (which means their gap to the function value at any point goes to zero as the branching hyper-rectangles 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 lead to finite convergence if there is some special structure on the optimal solutions such as an extreme point property [SS98, AS00].

Remark 1. The sBB algorithm developed here can be modified to accommodate separable PLFs in constraints using similar arguments as the classical results by Soland [Sol71]. 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.

2.2 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. Our branching rules are adopted from literature and explained later in § 4.1. In the remainder of this section, we outline our convex relaxation.

The convex envelope of a finite-valued function over a compact convex set is defined as the pointwise supremum of all the convex underestimators of that function over the set. Minimising

the function over the set is equivalent to minimising its convex envelope. However, this envelope is generally intractable to compute and the same is true for nonconvex PLFs also. 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 $S \cap H$. Since 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 $\mathsf{cvx}_H F(x) = \sum_{i=1}^n \mathsf{cvx}_{H_i} f_i(x_i)$. Each $\mathsf{cvx}_{H_i} f_i$ is a PLF but since 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 due to convex functions being u.s.c. over polytopes. Let us denote this underestimator for each i by $\mathsf{covx}_{H_i} f_i$. Summing these yields a convex continuous PLF underestimator on F,

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

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

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

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

where the second equality is from using the epigraph modelling step. Since each $\mathsf{covx}_{H_i} f_i$ is a convex continuous PLF, its epigraph is a polyhedron and so $\mathsf{covx}_{H_i} f_i$ is equal to the pointwise 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

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

Our construction of $\mathsf{covx}_{H_i} f_i$ 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:

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

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

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

A salient feature of this work is the efficient computation of the underestimator $\mathsf{covx}_{H_i} f_i$ and this is presented in Algorithm 1. However, this only represents the root node relaxation. In the search tree of sBB, H is successively partitioned into a sequence of hyper-rectangles $H^t \subset H$

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 $\mathsf{covx}_{H_i^t} f_i$ 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 hyper-rectangle 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

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

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 due to separability of the functions. The crucial thing though is whether $\mathsf{covx}_{H_{i_t}^t} f_{i_t}$ needs to be computed from scratch using the breakpoints of f_{i_t} and employing Algorithm 1 for univariate PLFs. This is not necessary because of a property of PLFs that only a subset of the breakpoints of $\mathsf{covx}_{H_{i_t}^t} f_{i_t}$ are different than those of $\mathsf{covx}_{H_{i_t}^s} f_{i_t}$, as we show in § 3.1. This allows for (on average) a quick and fast update to the underestimator of f_{i_t} over $H_{i_t}^s$ (assuming 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 $\underline{v}(H^t)$.

2.3 Relation to MILP and SOS2 approaches

It is well-known that all 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 loose this guarantee of providing the same bound as (4). In fact, only the Incremental and SOS2 model share this property called *hereditary sharpness* [HV22]. This property is very desirable since it leads to balanced search trees [YV13]. Indeed, experiments indicate that the *Incremental* and *SOS2* model perform very well on PLFs with a small number of segments and are only outperformed by the *Logarithmic* model with growing number of segments [cf. Reb16, HV22]. These considerations have been summarized by Huchette and Vielma [HV22] 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 the convex envelope 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 $\mathsf{covx}_{H_i} f_i$. 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 $\sum_{i=1}^{n} \left(2 \cdot \lceil \log_2(K_i - 1) \rceil + 3\right)$ constraints [cf. VAN10]. In contrast, the sBB relaxation (4) adds n continuous variables, 0 integers and $\sum_{i=1}^{n} E_i$ constraints. Since $K_i \gg 1$ 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 due to K_i being arbitrary.

Furthermore, the sBB offers the advantage of a sparser constraint matrix. Rebennack [Reb16] 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 non-zeros. Indeed, LPs with convex PLFs can be solved very efficiently by exploiting its structure [Fou85, Gor22]. Finally, spatial branching offers a higher degree of flexibility in branching decisions compared to integer or SOS2 branching. Both integer and spatial branching choose a branching variable x_i. However, while 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.

3 Univariate PLFs

It was outlined in § 2.2 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 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 utilised 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 $y^{k,+}$. For the left (resp. right) endpoint, we set the left (resp. right) limit to the function value. Thus, f is completed defined by the following finite collection of tuples as input for every $k \in \mathcal{K}_i$ we have as input the tuple

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

Define the following function 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 3.1. *$\underline{f}$ is a 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 $x \notin B_f$ since f is a PLF.

At any breakpoint b^k, we have

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

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

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

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

where cvx_I denotes the convex envelope operator over I. We use the following technical results to establish several properties of $\operatorname{covx}_I f$.

Lemma 3.2 (cf. Tuy [Tuy16, 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 3.3. *A continuous PLF in $\mathbb{R}^2$ 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* (resp. concave) *triangle* when the point in between lies below (resp. above) the segment joining the other two points.

Lemma 3.4. *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 3.3. Therefore, nonconvexity means 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 due to 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 $(i-1, i, i+1)$. □

Proposition 3.5. $\operatorname{covx}_I f$ *is a convex and continuous PLF underestimator of f whose breakpoints are given by the set*

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

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

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

Proof. Since $\mathsf{covx}_I f$ is the convex envelope of the PLF $\underline{f}$, it is obviously a convex PLF over I. Lemma 3.1 implies that this PLF is an underestimator of f. The convex envelope of a l.s.c. function is l.s.c. convex and is continuous over the interior of its domain and can only be discontinuous on the boundary. Combining this fact with Lemma 3.2, where we use I being a polyhedron in $\mathbb{R}$, gives us that $\mathsf{covx}_I f$ is a convex and continuous underestimator.

The breakpoints of $\mathsf{covx}_I f$ must be breakpoints of $\underline{f}$, and hence of f. The convex continuous PLF $\mathsf{covx}_I f$ is formed by joining its finitely many breakpoints. From Lemma 3.4 the characterisation of the breakpoints of the underestimator follows immediately. The breakpoints of a PLF form what is more generally called the generating set in global optimization literature for general nonconvex functions, and it is known that the 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. □

Another convex underestimator to f is the convex envelope of f, denoted by $\mathsf{cvx}_I f$. 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 $\mathsf{cvx}_I f(b^k) \geqslant \underline{f}(b^k)$ for $k \in \{0, K\}$. It is also not hard to see that $\mathsf{covx}_I f$ and $\mathsf{cvx}_I f$ have the same set of breakpoints. Therefore,

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

Thus, the only difference between $\mathsf{covx}_I f$ and $\mathsf{cvx}_I f$ is in their values at the endpoints where the latter will be u.s.c. due to Lemma 3.2 but may not be l.s.c..

We now build upon the characterisation of breakpoints in Proposition 3.5 to derive an efficient algorithm for computing $\mathsf{covx}_I f$ given f as an input through its breakpoints.

Proposition 3.6. *Algorithm 1 produces* $\mathsf{covx}_I f$ *after* $O(K)$ *iterations.*

Proof. Each application of the while loop is repeatedly checking the necessary and sufficient conditions for the slopes from Lemma 3.4. Furthermore, due to 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 k^{th} 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 $\mathsf{covx}_I f$ since they were obtained by checking the conditions in Lemma 3.4 and so correspond to the characterisation in Proposition 3.5. □

The running time of $O(K)$ for our algorithm is the best possible in the worst-case since a convex f would take K iterations due to 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 literature [cf. Cor+09, chap. 33.3] for generating the convex hull of a finite set of points in $\mathbb{R}^2$ (note that this convex hull is comprised 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 3.4 to discard points that will not be breakpoints of the envelope. 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.

Algorithm 1: Generating a convex continuous underestimator to a discontinuous PLF

Data: Lists $B = \{b^0, b^1, \dots, b^K\}$ 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 $\mathsf{covx}_I\, f : [l, u] \to \mathbb{R}$.

Compute $\overline{y}^k = \min\{y^k,\, y^{k,-},\, y^{k,+}\}$ for $k = 0, 1, \dots, K$

Initialize $\mathcal{B} = \{b^0\}$ and $\mathcal{Y} = \{\overline{y}^0\}$

for $k = 1$ **to** K **do**

  while $|\mathcal{B}| \geqslant 2$ ***and*** $\dfrac{Y[k] - \mathcal{Y}[-1]}{B[k] - \mathcal{B}[-1]} < \dfrac{\mathcal{Y}[-1] - \mathcal{Y}[-2]}{\mathcal{B}[-1] - \mathcal{B}[-2]}$ **do**

    Remove 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 $\mathcal{Y}$

3.1 Updating over subintervals

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

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

Of course, Algorithm 1 can be used to compute $\mathsf{covx}_{I'}\, f$, 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 1 to calculate $\mathsf{covx}_{I'}\, f$ requires rescanning all breakpoints of f in I'. Especially for PLFs with many segments, this can be expensive computation. However, this is usually not necessary since we show that $\mathsf{covx}_{I'}\, f$ equals $\mathsf{covx}_I\, f$ 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 breakpoint 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

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

Note that if b^{lo} and b^{up} do not exist then the updated envelope is trivial. Henceforth, assume 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 3.7. *Assume b^{lo} and b^{up} exist. The underestimator over I' can be described as follows:*

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

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 3.3 and 3.4. Since the breakpoints of f in I^1 were not breakpoints of $\mathsf{covx}_I\, f$, they form a concave triangle with the breakpoints in $\overline{I}^2$, 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. □

3.2 An Illustrative Example

The PLF in Figure 1 has 5 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 $(1, 3)$, $(3, 5)$, $(7, 2, 1, 3)$, $(8, 5)$, $(11, 7)$, $(13, 7)$.

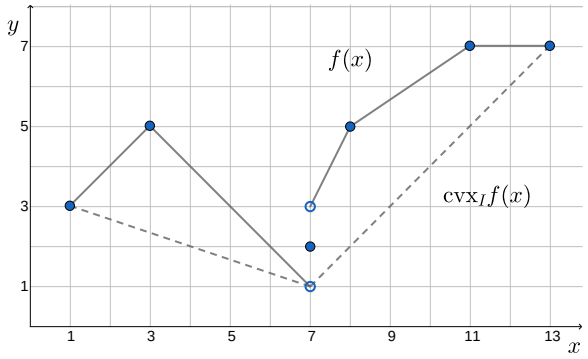


Figure 1: PLF $f(x)$ with discontinuity at 7 as solid line and convex envelope $\mathsf{cvx}_I\, f(x)$ over domain $I = [l, u] = [1, 13]$ as dashed line.

Applying Algorithm 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 $\mathsf{covx}_I\, f(x)$ formed by the tuples $(1, 3)$, $(7, 1)$ and $(13, 7)$ which equals $\mathsf{cvx}_I\, f(x)$ depicted in the figure. If Algorithm 1 is invoked to compute $\mathsf{covx}_{I'}\, f$ over $I' = [7, 13] \subset [1, 13]$, the input lists are $B = [7, 8, 11, 13]$ and $Y = [2, 5, 7, 7]$. Realize that $Y[0] = 2 \neq 1$ since the discontinuity at $b^0 = 7$ is at the edge of I' and hence $\overline{y}^0 = \mathtt{min}\{2, 2, 7\}$ since $y^{0,-} = y^0$.

Let $I = [1, 13]$ and $I' = [3, 10]$. $\mathsf{covx}_I\, f$ 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, $\mathsf{covx}_{I^1}\, \varphi$ is formed by $(3, 5)$, $(7, 1)$ and $\mathsf{covx}_{I^3}\, \varphi$ by $(7, 1)$,$(10, 6\frac{1}{3})$. Finally, $\mathsf{covx}_{I'}\, f$ is given by $(3, 5)$, $(7, 1)$ and $(10, 6\frac{1}{3})$.

This example illustrates that Proposition 3.7 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 3.7 is particularly useful for PLFs that accurately approximate a highly nonlinear function.

3.3 Lipschitz continuity

We close this section on univariate PLFs by noting two key properties. The first one is that when these functions are continuous they are also Lipschitz continuous with a constant equal to the largest magnitude of the slopes of the linear segments. This may perhaps be a known fact but since we could not find an explicit reference we present a proof, which uses elementary arguments.

Proposition 3.8. *If f is continuous over I, then it has a Lipschitz constant $L = \max_{k=0,1,\dots K-1}|m_k|$, where $m_k := \frac{y^{k+1}-y^k}{b^{k+1}-b^k}$ is the slope between breakpoints b^k and b^{k+1}.*

Proof. Take any $x, x' \in I$ with $x' \in [b^k, b^{k+1}]$ and $x \in [b^j, b^{j+1}]$ for some $k \leqslant j$. The case $k = j$ is trivial due to linearity in each piece, so assume $k < j$. We have

$$\begin{aligned} f(x) - f(x') &= \left[f(x) - f(b^j)\right] + \left[f(b^j) - f(b^{j-1})\right] + \cdots + \left[f(b^{k+1}) - f(x')\right] \\ &= m_j(x - b^j) + m_{j-1}(b^j - b^{j-1}) + \cdots + m_k(b^{k+1} - x') \\ &\leqslant \left[\max_{i=k,\dots,j} m_i\right] \left(x - b^j + b^j - b^{j-1} + \cdots + b^{k+1} - x'\right) \\ &= \left[\max_{i=k,\dots,j} m_i\right] (x - x') \\ \implies \left|f(x) - f(x')\right| &\leqslant \left[\max_{i=k,\dots,j} |m_i|\right] \left|x - x'\right| \leqslant L \left|x - x'\right|. \end{aligned}$$

□

The second property, which is argued in the next subsection, is that when a continuous PLF is used to approximate a univariate Lipschitz function then there is a formula for the number of uniformly spaced breakpoints required so that the PLF does not exceed a given approximation error.

3.3.1 Approximation of Lipschitz functions

Suppose we are given a L-Lipschitz univariate function $f : I \to \mathbb{R}$ on the interval $I := [l, u] \subset \mathbb{R}$. For every integer $k \geqslant 0$ there is a unique continuous PLF, which we denote by $g_k : I \to \mathbb{R}$, that approximates f through interpolation with $k+2$ uniformly spaced breakpoints which are given by $\{b_i := l + i\Delta(k) \colon i = 0, 1, \dots, k+1\}$ where $\Delta(k) := (u - l)/(k + 1)$ is the uniform spacing. The $k + 1$ segments of g_k are obtained by joining consecutive points so that the i^{th} segment joins the points (b_i, h_i) and (b_{i+1}, h_{i+1}) for $h_t := h(b_t)$. The error of a PLF with respect to f is defined as the largest approximation gap over the domain. Since the PLF interpolation is uniquely determined by the integer k, we can write the error function as

$$\xi(k) := \max_{x \in I} \left|h(x) - g_k(x)\right| = \max_{i=0,\dots,k} \max_{x \in [b_i, b_{i+1}]} \left|h(x) - [m_i x + h_i - m_i b_i]\right|, \tag{8}$$

where $m_i := (h_{i+1} - h_i)/(b_{i+1} - b_i)$ in the second equality comes from using the definition of g_k in each subinterval. Straightforward arguments lead to a formula for the number of breakpoints required to achieve a given error.

Proposition 3.9. *For any $\varepsilon > 0$, a continuous PLF approximation of f has the same Lipschitz constant L, and it has an error at most ε when c/ε many uniformly spaced breakpoints are used, where $c = \lceil 2L(u - l)\rceil$.*

Proof. The first assertion about the Lipschitz constant follows directly from Proposition 3.8. For the second claim, we need the following observation on Lipschitz continuity.

Lemma 3.10. *Let $h_1 : X \to \mathbb{R}$ be L_1-Lipschitz and $h_2 : X \to \mathbb{R}$ be L_2-Lipschitz on a closed set $X \subset \mathbb{R}^n$. Then $h_1 - h_2 : x \in X \mapsto h_1(x) - h_2(x)$ and $h_1 + h_2 : x \in X \mapsto h_1(x) + h_2(x)$ are $L_1 + L_2$-Lipschitz on X.*

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

$$\begin{aligned}\big\|(h_1 - h_2)(x) - (h_1 - h_2)(y)\big\| &= \Big\|h_1(x) - h_1(y) - \big(h_2(x) - h_2(y)\big)\Big\| \\ &\leqslant \big\|h_1(x) - h_1(y)\big\| + \big\|h_2(x) - h_2(y)\big\| \\ &\leqslant L_1\|x - y\| + L_2\|x - y\| \\ &= (L_1 + L_2)\|x - y\|\,,\end{aligned}$$

where the first inequality is the triangle inequality for norms, and the second inequality is Lipschitz continuity of h_1 and h_2. Similar derivation works for the sum. □

Consider the function $h_i(x) := h(x) - g_k(x) = h(x) - m_i x - h_i + m_i b_i$, for $x \in [b_i, b_{i+1}]$, that appears in the error function $\xi(k)$. From Lemma 3.10, h_i has a Lipschitz constant of $L + |m_i|$. Note that $h_i(b_i) = 0$ due to exactness of g_k at breakpoints. Hence, for any $x \in [b_i, b_{i+1}]$, the definition of Lipschitz continuity for h_i gives us $\big|h_i(x)\big| \leqslant (L + |m_i|)(x - b_i) \leqslant (L + |m_i|)\Delta(k)$, where the last inequality is due to uniform spacing between breakpoints. For m_i we have $|m_i| = \frac{|h_{i+1} - h_i|}{\Delta(k)} \leqslant L$, where the inequality is due to Lipschitz continuity of f. Hence, $\big|h_i(x)\big| \leqslant 2L\,\Delta(k)$. Since $\xi(k) = \max_{i=0,\ldots,k} \max_{x \in [b_i, b_{i+1}]} \big|h_i(x)\big|$, it follows that $2L\,\Delta(k) \leqslant \varepsilon$ is a sufficient condition for $\xi(k) \leqslant \varepsilon$, and then using the definition of $\Delta(k) := (u - l)/(k + 1)$ implies that k must be at least $(2L(u - l)/\varepsilon) - 1$. □

4 Spatial branch-and-bound algorithm

Our main ideas for an sBB algorithm to solve the PLF optimization problem (1) were sketched in § 2.2. The algorithm is presented formally in Algorithm 2. The bounding operation is specified next and the branching schemes in § 4.1. Convergence is discussed in § 4.2.

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

$$\mathcal{P}^k \;:\quad v(H^k) = \inf_x \; F(x) \quad \text{s.t.} \quad x \in S \cap H^k. \tag{9a}$$

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

$$\mathcal{R}^k \;:\quad v(H^k) \geqslant \underline{v}(H^k) = \min_x \; \operatorname{covx}_{H^k} F(x) \quad \text{s.t.} \quad x \in S \cap H^k, \tag{9b}$$

where the underestimator is defined as

$$\operatorname{covx}_{H^k} F(x) = \sum_{i=1}^{n} \operatorname{covx}_{H_i^k} f_i(x_i). \tag{9c}$$

Since $\operatorname{covx}_{H^k} F$ is polyhedral as per the results of the previous section, using the epigraph modelling 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

$$\operatorname{covx}_{H^k} F(x) = \operatorname{covx}_{H_j^k} f_j(x_j) + \sum_{i \neq j} \operatorname{covx}_{H_i^k} f_i(x_i). \tag{9d}$$

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 (9d) gives us

$$\operatorname{convx}_{H^k} F(x) = \operatorname{convx}_{H^k_{i_k}} f_{i_k}(x_{i_k}) + \sum_{i \neq i_k} \operatorname{convx}_{H^p_i} f_i(x_i). \tag{9e}$$

Note that when using Proposition 3.7 to update the underestimator over a child node, the breakpoints of $\operatorname{convx}_{H^k} F$ 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 1 can be called at each child node H^l to compute $\operatorname{convx}_{H^l} F$ from scratch and no additional data need to be stored.

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 optimal solution and optimal value of relaxation $\mathcal{R}^k$, α^k and β^k are the global upper and lower bound, 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.

Algorithm 2: Spatial branch-and-bound algorithm for PLF optimization

Root node: Compute $\operatorname{convx}_H F$ as per **??** using Algorithm 1 for $\operatorname{convx}_{H_i} f_i$ 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 a $H^{l^*} \in \arg\min\{r^l : H^l \in \mathcal{L}\}$. 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 § 4.1. Let x_{i_k} denote the branching variable
  Bounding: for $l \in \{1, 2\}$ **do**
    Compute $\operatorname{convx}_{H^{k,l}} F$ as per (9e) with $p = k$ and $k = k, l$ and using Proposition 3.7 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: $\mathcal{L} \leftarrow \left(\mathcal{L} \setminus \{H^k\}\right) \cup \mathcal{H}$
  Pruning: Fathom subproblems by bound dominance as $\mathcal{L} \leftarrow \mathcal{L} \setminus \{H^l : r^l \geqslant \alpha^k - \varepsilon\}$.
end

4.1 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 2 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 [FS69] and variations of it can be found for instance in the solver BARON [TS04]. 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 [Adj+98].

Largest-error branching rule: Select the index which contributes most to the convexification gap at x^k by

$$\tau \in \operatorname*{arg\,max}_{i=1,\dots,n} \left[f_i(x_i^k) - \operatorname{covx}_{H_i^k} f_i(x_i^k) \right], \tag{10a}$$

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

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

Longest-edge branching rule: Select the index with the longest edge by

$$\tau \in \operatorname*{arg\,max}_{i=1,\dots,n} u_i^k - l_i^k, \tag{11a}$$

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

$$H^{k,1} = \left\{ x \in H^k : x_\tau \leqslant \frac{u_\tau^k - l_\tau^k}{2} \right\} \quad \text{and} \quad H^{k,2} = \left\{ x \in H^k : x_\tau \geqslant \frac{u_\tau^k - l_\tau^k}{2} \right\}. \tag{11b}$$

Breakpoint branching rule: Select the index τ by the largest-error rule (10a) applied only to breakpoints, i.e. select a breakpoint b_τ^* with the largest error. Partition H^k at this breakpoint,

$$H^{k,1} = \left\{ x \in H^k : x_\tau \leqslant b_\tau^* \right\} \quad \text{and} \quad H^{k,2} = \left\{ x \in H^k : x_\tau \geqslant b_\tau^* \right\}. \tag{12}$$

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. YV13].

4.2 Convergence guarantees

Falk and Soland [FS69, Theorem 2] established asymptotic convergence of the largest-error branching rule when F is continuous. 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., Theorem 1 in their paper 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 since we also consider a separable objective. Furthermore, as mentioned in § 2.1 and described in [LS13, chap. 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 4.1. *When each f_i is l.s.c., Algorithm 2 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 $\operatorname{covx}_I f$ is exact at each breakpoint. 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. Since there are finitely many breakpoints, the claim follows because all feasible leaf nodes of the sBB tree will yield an exact representation of the original problem (1). □

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 [FS69] for general separable functions.

Proposition 4.2. *The largest-error branching rule converges in the limit for continuous PLFs.*

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^+.

(ii) only one index $\tau \in \mathcal{I}$ gets branched on infinitely often.

Since we are only interested in the limit behaviour, we can therefore focus exclusively on the index τ. First, note that f_τ and thus also $\operatorname{covx}_{H_\tau^q} f_\tau$ is Lipschitz-continuous with constant L_τ for all iterations q. Now, let us define function $\psi_\tau^q(x_\tau) = f_\tau(x_\tau) - \operatorname{covx}_{H_\tau^q} f_\tau(x_\tau)$ over H_τ^q. Note that ψ_τ^q is Lipschitz with constant $2L_\tau$. By the largest-error branching rule, namely (10b), we obtain $x_\tau^{q-1} \in \operatorname{bd}(H_\tau^q)$ and thus $x_\tau^q, x_\tau^{q-1} \in H_\tau^q$. Consequently,

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

Since f_τ is continuous and $x_\tau^{q-1} \in \operatorname{bd}(H_\tau^q)$, we obtain by **??** that $\psi_\tau^q(x_\tau^{q-1}) = 0$ and hence

$$\left|f_\tau(x_\tau^q) - \operatorname{covx}_{H_\tau^q} f_\tau(x_\tau^q)\right| \leqslant 2L_\tau \cdot \left|x_\tau^q - x_\tau^{q-1}\right| .$$

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

$$\lim_{q\to\infty} \left|f_\tau(x_\tau^q) - \operatorname{covx}_{H_\tau^q} f_\tau(x_\tau^q)\right| = 0 .$$

Finally, there is a $\overline{q}$ so that for all $q > \overline{q}$ the branching index τ is selected by (10a). Hence, we get that

$$\forall i \in \mathcal{I} \setminus \{\tau\} : \lim_{q\to\infty} \left(f_i(x_i^q) - \operatorname{covx}_{H_i^q} f_i(x_i^q)\right) = 0 \,. \tag{13a}$$

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

$$\lim_{q\to\infty} \alpha^q \leqslant \lim_{q\to\infty} F(x^q) = \lim_{q\to\infty} \operatorname{covx}_{H^q} F(x^q) = \lim_{q\to\infty} r^q = \lim_{q\to\infty} \beta^q. \tag{13b}$$

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 $\psi_\tau^p(x_\tau^p) = f_\tau(x_\tau^p) - \operatorname{covx}_{H_\tau^p} f_\tau(x_\tau^p) = 0$. By similar arguments like in (13a) and (13b) follows then $\beta^p = \alpha^p$ 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 literature [cf. TH88, Theorem 2.3], i.e., $\lim_{k\to\infty} \beta^k = v^* = \lim_{k\to\infty} \alpha^k$ and every accumulation point of $\{\overline{x}^k\}$ 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 $\lim_{q\to\infty} \alpha^q = \lim_{q\to\infty} \beta^q$. □

Wechsung and Barton [WB14] 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 and 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.

5 Computational experiments

We compare the computational performance of the sBB algorithm with MILP approaches from literature. In § 5.1, we consider network flow problems with concave PLFs and in § 5.2 knapsack problems with both nonconcave and concave PLFs. Details on our experiments, including performance profiles and timing statistics, are in § 5.3 and a discussion of our numerical results is in § 5.4.

Let us begin by outlining the design of our experiments. Algorithm 2 was implemented in Python version 3.9. The largest-error branching rule is chosen because in our initial testing it seemed to do better than the other rules described in § 4.1. Nodes were selected using the best-bound rule. The LPs on the nodes are solved with Gurobi version 9.5. The MILP models are generated in Julia version 1.7 using the package *PiecewiseLinearOpt* developed by Huchette and Vielma [HV22] and are solved by Gurobi version 9.5 with standard settings. Similar to findings in literature [cf. VAN10], first experiments indicated that linear-sized MILP models and the SOS2 branch-and-cut procedure inside Gurobi are not competitive to logarithmic-sized models when nonconvex PLFs with 50 or more segments are involved. To avoid excessive computation times due to vast timeouts, we therefore compare only against the 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) [VAN10], and the recently introduced *Binary Zig-Zag* (ZZB) and *General Integer Zig-Zag* (ZZI) models [HV22]. All tests were carried out on a server with 3.70GHz and 128GB RAM. 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, the MILP generation as well as the instance generator are available at GitHub under `https://github.com/ThomasHubner/sBB_PiecewiseLinOpt.git`.

Instance generation is described in the following sections. We restricted to continuous PLFs since our sBB does not yet have a convergence guarantee in the presence of general discontinuities, and we believe that extensive experiments with continuous functions are enough to achieve the primary aim of this study which is to analyze the numerical performance with respect to scalability in the number of segments. Although convergence is achieved for l.s.c. PLFs using the longest-edge branching rule, as remarked at the end of the previous section, this rule works poorly in practice and so we do not expect it will outperform the MILP models. It is also worth noting that the only algorithm for general discontinuous PLFs provided by Farias Jr. et al. [FZZ08] is only available as a proof of concept and has never been fully implemented.

5.1 Network flow problem with concave cost

Network flow problems with nonconvex PLFs occur in many applications ranging from telecommunications to logistics [CGM07]. They can be defined as follows:

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

An instance of the network flow problem is created similar to [KFN06, VAN10, HV22] as follows. First, declare each node $i = 1,\dots,n-1$ 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 $\big(b_i^k, f(b_i^k)\big)$, $k = 0,\dots,K$ of the concave PLFs $f_i(x_i)$ are determined as follows: Set $b_i^0 = l_i = 0$ and $b_i^K = u_i \sim \text{Uniform}(5,50)$ and generate $K-1$ points $b_i^k \sim \text{Uniform}(l_i,u_i)$, $k = 1,\dots,K-1$ and order them. Subsequently, generate K slopes by $\text{slopes}_k \sim \text{Uniform}(1,2000)/1000$, $k = 1,\dots,K$ 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 $f_i(b_i^k) = \text{slope}_k \cdot (b_i^k - b_i^{k-1}) + f_i(b_i^{k-1})$, $k = 1,\dots,K$.

We perform our computational test on network flow problems with $n = 10$ nodes. For each K, 100 random network flow instances are generated and solved. The statistics of the solve times are given in Table 1. We display the median (med), the arithmetic mean (avg) and the standard deviation (std) 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).

5.2 Knapsack problem with approximated nonlinearities

As discussed in the introduction, 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:

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

Table 1: Solve times [s] for network flow problems with continuous concave PLFs.

Method	Med.	Avg.	Std.	Win	Fail	Method	Med.	Avg.	Std.	Win	Fail
a) 10 segments						d) 1,000 segments					
Log	**0.52**	0.60	0.32	36	0	sBB	**7.7**	16.7	27.2	98	0
ZZI	**0.52**	0.54	0.24	46	0	Log	**58.5**	65.5	38.4	1	0
ZZB	**0.64**	0.80	0.59	9	0	ZZI	**84.9**	78.8	31.0	1	0
DLog	**0.65**	0.76	0.41	9	0	ZZB	**91.7**	87.4	36.2	0	0
sBB	**7.23**	12.86	19.44	0	0	DLog	**96.5**	105.7	56.0	0	0
b) 100 segments						e) 5,000 segments					
ZZI	**3.62**	4.50	2.36	44	0	sBB	**13.5**	21.7	33.6	100	0
ZZB	**3.76**	4.25	2.20	35	0	Log	**434.0**	441.8	214.2	0	0
Log	**4.66**	5.89	3.66	10	0	ZZI	**631.0**	631.7	287.4	0	0
DLog	**7.19**	9.31	6.15	0	0	ZZB	**748.6**	786.4	357.4	0	1
sBB	**9.15**	14.13	16.51	11	0	DLog	**843.2**	909.9	459.8	0	6
c) 500 segments						f) 10,000 segments					
sBB	**9.7**	15.9	16.9	86	0	sBB	**16**	28	39	100	0
ZZB	**22.6**	26.8	13.0	7	0	Log	**940**	1,004	384	0	3
Log	**23.2**	28.1	15.4	6	0	ZZI	**1,496**	1,469	336	0	30
ZZI	**37.8**	35.0	14.8	1	0	ZZB	**1,818**	1,677	246	0	60
DLog	**49.2**	53.4	27.7	0	0	DLog	**1,825**	1,609	394	0	68

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. [Cas+03].

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, $K-1$ points $b_i^k \sim \text{Uniform}(l_i, u_i), k \neq \{0, K\}$ are generated and ordered. The first and last breakpoint is set to $b_i^0 = l_i$ and $b_i^K = u_i$. Each h_i is then approximated by a PLF f_i with K segments given by the breakpoints $\big(b_i^k, h_i(b_i^k)\big)$. The demand parameter d is then as well randomly determined by $d \sim \text{Uniform}\big(l + \frac{1}{4} \cdot (u-l),\ u - \frac{1}{4} \cdot (u-l)\big)$ 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, 100 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 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 the table *e.g.* $K = 20$ and $K+1 = 50$.

Table 2: Nonconvex univariate functions.

#	Function	Domain	#	Function	Domain
1	$e^{-3x-12} - x^2 + 20$	$[-5, 5]$	11	$x^4 - 12x^3 + 47x^2 - 60x$	$[-1, 7]$
2	$-0.2 \cdot e^{-x} + x^2$	$[-5, 5]$	12	$x^6 - 15x^4 + 27x^2 + 250$	$[-4, 4]$
3	$x^3 \cdot e^{-x^2}$	$[-5, 5]$	13	$x^4 - 10x^3 + 35x^2 - 50x + 24$	$[0, 5]$
4	$\frac{x^5 - 20x^2 + 5}{x^4 + 1}$	$[-10, 10]$	14	$0.2x^5 - 1.25x^4 + 2.33x^3 - 2.5x^2 + 6x$	$[-1, 4]$
5	$\log(3x) \cdot \log(2x) - 1$	$[0.1, 10]$	15	$x^3 - 7x + 7$	$[-4, 4]$
6	$10\log(x) - 3x + (x-5)^2$	$[0.1, 10]$	16	$\frac{(x^4-4x+10)}{(x^2+1)} - 1$	$[-5, 5]$
7	$\frac{-x^5 - 10x^2}{x^6 + 5}$	$[-10, 10]$	17	$-x^5 \cdot e^{(} - x^2)$	$[-10, 10]$
8	$x \cdot e^{-x^2}$	$[-5, 5]$	18	$x^5 - 3x^4 + 4x^3 + 2x^2 - 10x - 4$	$[-1.5, 3]$
9	$-\frac{x^7}{5040} + \frac{x^5}{120} - \frac{x^3}{3} + x$	$[-4, 4]$	19	$\frac{(x^3 - 5x + 6)}{(x^2 + 1)} - 1$	$[-5, 5]$
10	$\frac{x^2 - 5x + 6}{x^2 + 1} - 1$	$[-10, 10]$	20	$\frac{1}{x} + 2\log(x) - 2$	$[0.1, 10]$

Table 3: Solve times [s] for nonconcave knapsack problems.

Method	Med.	Avg.	Std.	Win	Fail	Method	Med.	Avg.	Std.	Win	Fail
a) 10 segments						d) 1,000 segments					
Log	**0.09**	0.19	0.95	38	0	sBB	**8.1**	73.0	222.6	65	0
DLog	**0.10**	0.11	0.07	4	0	Log	**14.2**	17.4	10.3	31	0
ZZI	**0.10**	0.10	0.05	24	0	ZZI	**30.6**	35.9	29.2	1	0
sBB	**0.12**	0.26	0.74	33	0	DLog	**34.5**	113.4	286.6	1	2
ZZB	**0.13**	0.13	0.07	1	0	ZZB	**35.6**	41.3	27.5	2	0
b) 100 segments						e) 5,000 segments					
sBB	**0.52**	3.92	12.89	60	0	sBB	**41.8**	186.4	361.4	79	2
Log	**0.72**	0.97	0.69	30	0	Log	**173.5**	421.4	526.9	11	8
ZZB	**1.02**	1.40	1.12	4	0	DLog	**225.3**	438.2	503.0	6	8
ZZI	**1.14**	1.35	0.91	5	0	ZZI	**257.6**	365.1	327.5	3	1
DLog	**1.42**	2.23	5.02	1	0	ZZB	**278.8**	363.4	299.3	1	1
c) 500 segments						f) 10,000 segments					
sBB	**3.9**	35.1	181.1	56	1	sBB	**107**	250	345	95	1
Log	**5.0**	6.0	4.1	39	0	ZZI	**663**	762	404	1	4
ZZI	**8.3**	9.7	10.1	3	0	ZZB	**675**	796	406	0	6
ZZB	**8.9**	9.9	8.0	2	0	DLog	**888**	971	612	2	25
DLog	**12.1**	22.1	33.4	0	0	Log	**899**	1,011	663	1	34

5.2.1 Concave knapsack problems

To evaluate the impact of non-concavity 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

Table 4: Relative improvement in real objective value over previous number of segments K. For $K = 20$ the improvement in real objective value is measured relative to the value of $K = 10$.

K	Min.	Med.	Avg.	Max.	Std.
20	-37.0 %	25.84 %	**80.93** %	861.08 %	172.48 %
50	-6.58 %	6.26 %	**12.08** %	81.73 %	16.44 %
100	-1.55 %	1.03 %	**2.16** %	11.96 %	2.74 %
500	0.939 ‰	5.006 ‰	**7.624** ‰	47.971 ‰	8.565 ‰
1,000	-0.128 ‰	0.273 ‰	**0.531** ‰	3.509 ‰	0.718 ‰
5,000	-0.006 ‰	0.076 ‰	**0.127** ‰	1.096 ‰	0.178 ‰
10,000	-0.004 ‰	0.003 ‰	**0.007** ‰	0.054 ‰	0.011 ‰

recomputed by using the new slopes and x-coordinate of the breakpoints. The table 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.

Table 5: Solve times [s] for concave knapsack problems.

Method	Med.	Avg.	Std.	Win	Fail	Method	Med.	Avg.	Std.	Win	Fail
a) 10 segments						d) 1,000 segments					
Log	**0.09**	0.20	0.95	51	0	sBB	**6.1**	180.4	418.5	70	3
ZZI	**0.11**	0.11	0.04	17	0	Log	**17.6**	60.2	94.3	29	0
DLog	**0.11**	0.12	0.06	9	0	DLog	**61.2**	269.9	454.1	0	4
ZZB	**0.12**	0.13	0.05	5	0	ZZB	**72.5**	243.7	403.0	0	2
sBB	**0.16**	0.23	0.24	18	0	ZZI	**76.2**	220.8	357.2	1	1
b) 100 segments						e) 5,000 segments					
sBB	**1.02**	2.95	5.10	45	0	sBB	**49.7**	442.4	691.7	79	18
Log	**1.09**	1.22	0.54	47	0	Log	**258.1**	648.7	698.6	3	24
ZZI	**1.84**	2.03	1.11	7	0	ZZI	**1,110.3**	1,137.6	631.7	0	38
ZZB	**1.90**	2.10	1.28	0	0	ZZB	**1,238.9**	1,165.6	650.0	0	44
DLog	**2.04**	2.06	0.96	1	0	DLog	**1,281.1**	1,060.7	760.4	0	45
c) 500 segments						f) 10,000 segments					
sBB	**2.9**	72.7	246.0	67	1	sBB	**22**	466	733	78	22
Log	**6.7**	14.6	29.1	31	0	Log	**502**	904	706	0	33
ZZB	**16.0**	51.1	126.2	0	0	DLog	**1,309**	1,168	658	0	46
ZZI	**17.0**	44.8	103.6	1	0	ZZI	**1,818**	1,520	479	0	65
DLog	**17.7**	41.1	77.3	1	0	ZZB	**1,819**	1,474	481	0	59

5.3 Details on Computational Experiments

This section dives deeper into our numerical results. Means and medians are point estimators that don't 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 behaviour of the different models and algorithms by plotting the performance profiles of their solution times. Such profiles are a standard tool in computational optimization. We also investigate the amount of time that the sBB spends on its different operations.

5.3.1 Performance profiles

Each model/algorithm gets one profile curve which is interpreted as its approximate cumulative distribution function, and the curve that is in the top left corner in each figure has stochastic dominance over other curves and hence corresponds to the best method. In particular, any point with coordinates (a, b), where $a \in [0, 1]$ and $b \in \{0, \ldots, 100\}$, on a curve means that for each of the b many instances the solution time for the method was no greater than $(1-a)T_I^{\min} + aT_I^{\max}$, where $T_I^{\min}$ and $T_I^{\max}$ are the minimum and maximum times to solve instance I across all methods. The vertical intercept ($a = 0$) gives the number of instances for which that method solved the fastest (the Win column in previous tables), and as $a \to 1$ the method is solving slower than others.

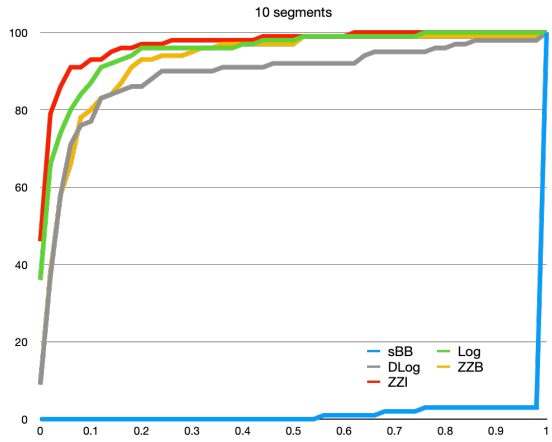
Figures 2 and 3 give these profiles, respectively, for the network flow problems and knapsack problems with concave PLFs. In the former, the sBB profiles are consistent with Table 1 and give superior performance for 500 segments and beyond. The profiles for the concave knapsacks reveal that for up to 1000 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 same number of instances as Log and dominates DLog and the two zig-zag models, whereas beyond 500 segments, the dominance of sBB keeps growing steadily. Similar behaviour is observed for the nonconcave 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.

5.3.2 Timing statistics for the sBB

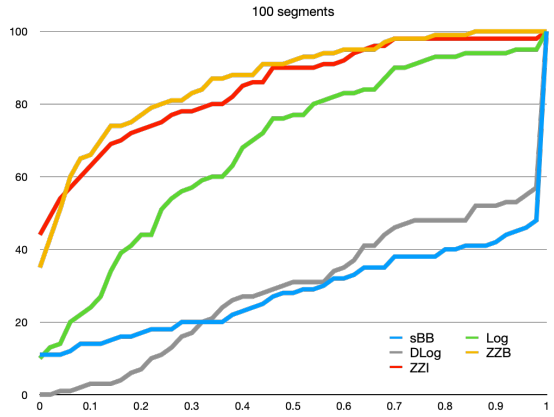
Here, we take a look at some details of the operation of the sBB implementation. Table 6 indicates that solving LPs take only a small share of the sBBs solution time although it is by far the most complicated operation in a branch-and-bound algorithm. Instead, operations like building the model and repeatedly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

Table 6: Average proportion of runtime that is allotted to the various sub-operations of the sBB algorithm when solving knapsack problems.

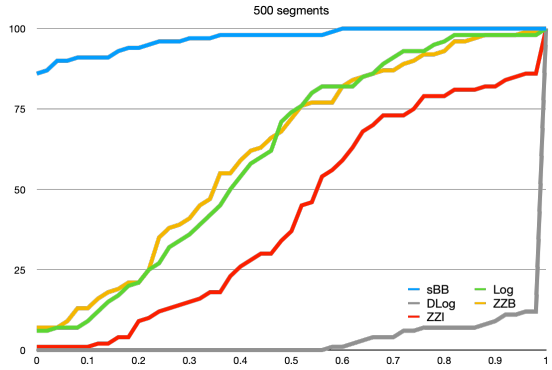
Operation	Nonconcave		Concave	
	$K = 10$	$K = 10,000$	$K = 10$	$K = 10,000$
Gurobi Interface	31%	55%	15%	8%
Solving LPs	6%	30%	2%	2%
Envelope Generation	3%	3%	1%	50%
PLF Evaluations	57%	1%	78%	35%
Other Operations	3%	11%	4%	5%



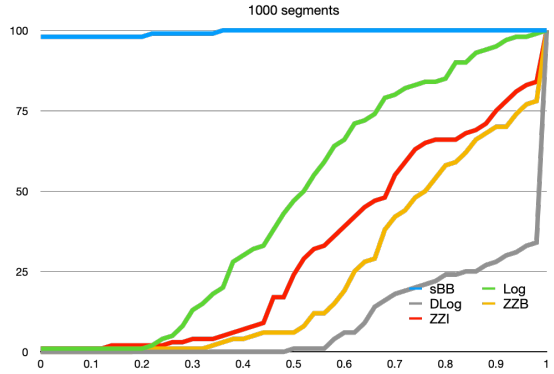
(a) 10 segments



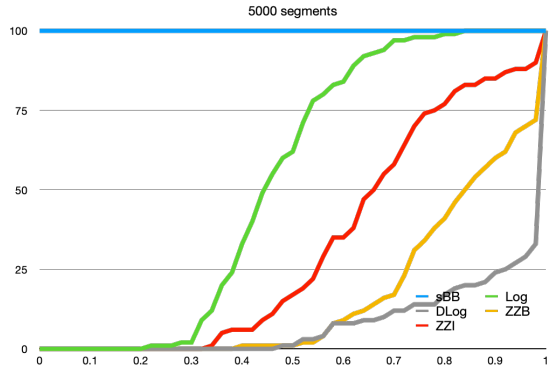
(b) 100 segments



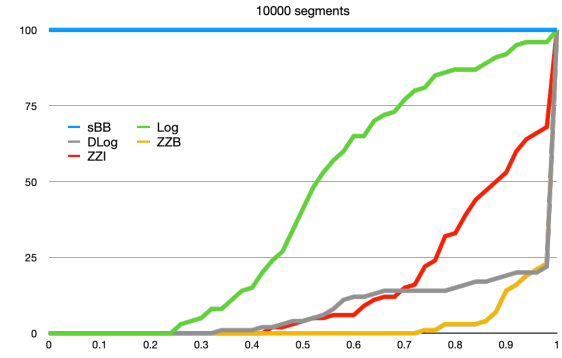
(c) 500 segments



(d) 1,000 segments



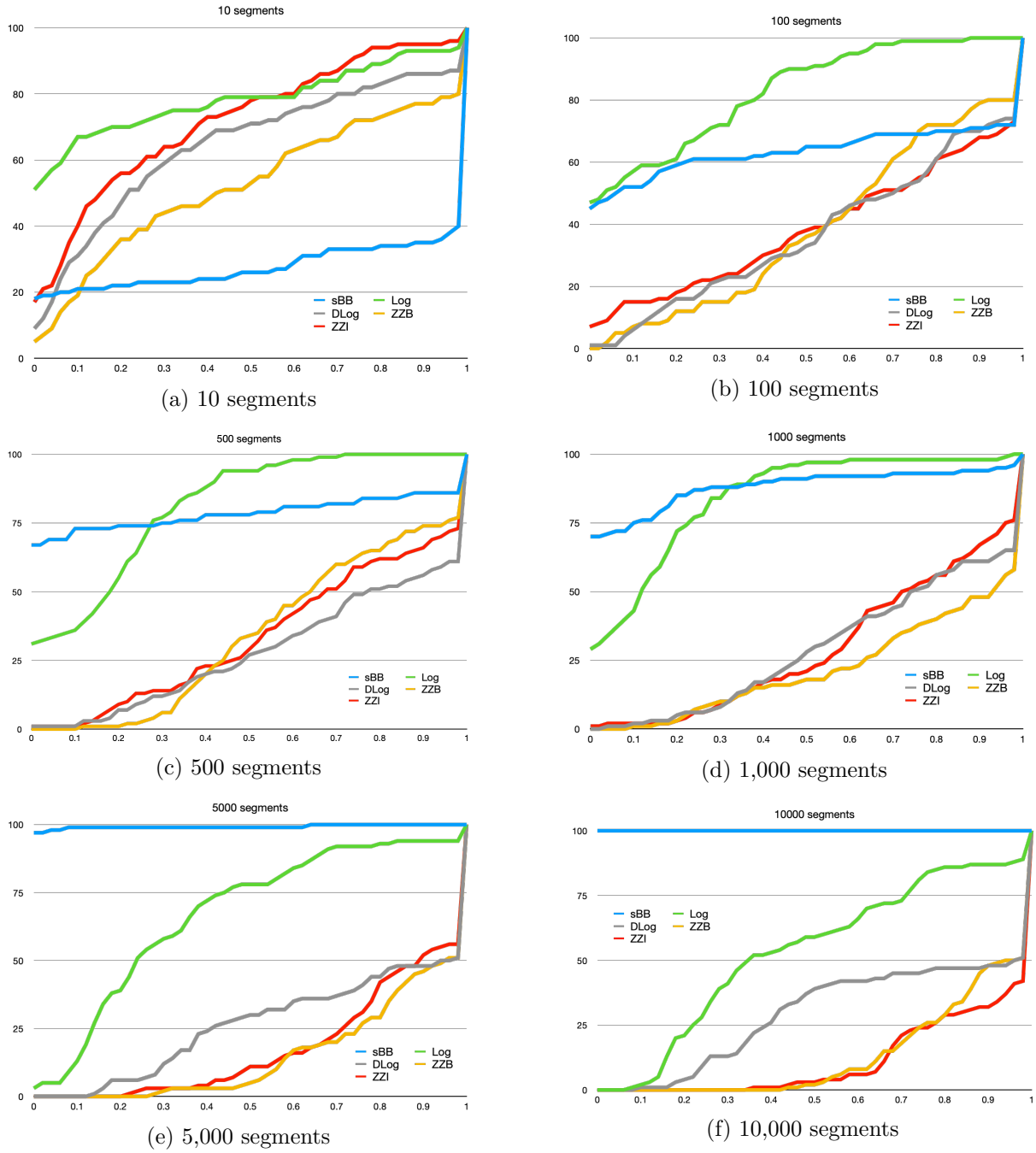
(e) 5,000 segments



(f) 10,000 segments

Figure 2: Performance profiles for network flow PLF problems.

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 1 which is always entered since 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 breakpoint of the PLF without entering any loop.



(a) 10 segments

(b) 100 segments

(c) 500 segments

(d) 1,000 segments

(e) 5,000 segments

(f) 10,000 segments

Figure 3: Performance profiles for concave PLF knapsack problems.

5.4 Discussion

Due to the difference in implementation quality — a rudimentary sBB implementation in Python compared to 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 to logarithmic approaches.

This is further illustrated in *performance profiles* given in § 5.3.1. 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 § 2.3). 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 § 2.3, the Incremental and SOS2 models, which guarantee sharpness in the entire search tree, usually outperform logarithmic models for problems with few segments. Since 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, since spatial branching can additionally lead to more balanced search trees by branching at the previous solution instead of at the breakpoints (see § 4.1). However, the computational results do not support this claim. We believe that the poor performance of the sBB compared to logarithmic approaches on problems with few segments is due to 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 SOS2 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.

6 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 due to its slim and sparse LP relaxations. However, for PLFs with few segments, MILP models such as the classical Incremental model might be faster due to their large formulation and the thus 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 towards 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 non-separable 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 paragraph.

The ideas of pseudocost, strong and reliability branching from MILP [AKM05] could be adopted here. Secondly, there have been many works [Ben90, Kes+04, DAm+20] 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. Thirdly, 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. (barring a special case). This does not seem to be an easy task since 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 since the SOS2 branching rule has been generalized [FZZ08], although only as a proof-of-concept and not something that has been implemented in MILP solvers. Lastly, one could explore machine learning techniques for branching decisions, as was done recently for nonconvex polynomial optimization problems [Gha+23].

Acknowledgements The research of the third author is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) [Grant 445857709].

Bibliography

[AKM05] T. Achterberg, T. Koch, and A. Martin. "Branching rules revisited". In: *Operations Research Letters* 33.1 (2005), pp. 42–54.

[AGX19] W. Adams, A. Gupte, and Y. Xu. "Error bounds for monomial convexification in polynomial optimization". In: *Mathematical Programming* 175 (2019), pp. 355–393.

[Adj+98] C. S. Adjiman, S. Dallwig, C. A. Floudas, and A. Neumaier. "A global optimization method, αBB, for general twice-differentiable constrained NLPs – I. Theoretical advances". In: *Computers & Chemical Engineering* 22.9 (1998), pp. 1137–1158.

[Bär+23] A. Bärmann, R. Burlacu, L. Hager, and T. Kleinert. "On piecewise linear approximations of bilinear terms: structural comparison of univariate and bivariate mixed-integer programming formulations". In: *Journal of Global Optimization* 85.4 (2023), pp. 789–819.

[BHH22] B. Beach, R. Hildebrand, and J. Huchette. "Compact mixed-integer programming formulations in quadratic optimization". In: *Journal of Global Optimization* 84 (2022), pp. 869–912.

[BF76] E. Beale and J. J. Forrest. "Global optimization using special ordered sets". In: *Mathematical Programming* 10.1 (1976), pp. 52–69.

[Ben90] H. P. Benson. "Separable concave minimization via partial outer approximation and branch and bound". In: *Operations Research Letters* 9.6 (1990), pp. 389–394.

[BGS20] R. Burlacu, B. GeiSSler, and L. Schewe. "Solving mixed-integer nonlinear programmes using adaptively refined mixed-integer linear programmes". In: *Optimization Methods and Software* 35.1 (2020), pp. 37–64.

[Cas+03] L. G. Casado, J. A. Martínez, I. García, and Y. D. Sergeyev. "New interval analysis support functions using gradient information in a global minimization algorithm". In: *Journal of Global Optimization* 25.4 (2003), pp. 345–362.

[Cor+09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2009.

[CGM03] K. L. Croxton, B. Gendron, and T. L. Magnanti. "A comparison of mixed-integer programming models for nonconvex piecewise linear cost minimization problems". In: *Management Science* 49.9 (2003), pp. 1268–1273.

[CGM07] K. L. Croxton, B. Gendron, and T. L. Magnanti. "Variable disaggregation in network flow problems with piecewise linear costs". In: *Operations Research* 1 (2007), pp. 146–157.

[DAm+20] C. D'Ambrosio, J. Lee, D. Skipper, and D. Thomopulos. "Handling separable non-convexities using disjunctive cuts". In: *Combinatorial Optimization: ISCO 2020*, ed. by M. Baïou, B. Gendron, et al. Vol. 12176. Lecture Notes in Computer Science. Springer Cham, 2020, pp. 102–114.

[Dan60] G. B. Dantzig. "On the significance of solving linear programming problems with some integer variables". In: *Econometrica* 28.1 (1960), pp. 30–44.

[DG15] S. S. Dey and A. Gupte. "Analysis of MILP techniques for the pooling problem". In: *Operations Research* 63.2 (2015), pp. 412–427.

[FS69] J. E. Falk and R. M. Soland. "An algorithm for separable nonconvex programming problems". In: *Management Science* 15.9 (1969), pp. 550–569.

[Far+13] I. R. de Farias, E. Kozyreff, R. Gupta, and M. Zhao. "Branch-and-cut for separable piecewise linear optimization and intersection with semi-continuous constraints". In: *Mathematical Programming Computation* 5.1 (2013), pp. 75–112.

[FZZ08] I. R. de Farias Jr., M. Zhao, and H. Zhao. "A special ordered set approach for optimizing a discontinuous separable piecewise linear function". In: *Operations Research Letters* 36.2 (2008), pp. 234–238.

[Fou85] R. Fourer. "A simplex algorithm for piecewise-linear programming I: Derivation and proof". In: *Mathematical Programming* 33.2 (1985), pp. 204–233.

[FSB10] C. L. Frenzen, T. Sasao, and J. T. Butler. "On the number of segments needed in a piecewise linear approximation". In: *Journal of Computational and Applied mathematics* 234.2 (2010), pp. 437–446.

[Gei+12] B. GeiSSler, A. Martin, A. Morsi, and L. Schewe. "Using piecewise linear functions for solving MINLPs". In: *Mixed Integer Nonlinear Programming*, ed. by J. Lee and S. Leyffer. Vol. 154. IMA Volumes in Mathematics and its Applications. Springer, 2012, pp. 287–314.

[Gha+23] B. Ghaddar, I. Gómez-Casares, J. González-Díaz, B. González-Rodríguez, B. Pateiro-López, and S. Rodríguez-Ballesteros. "Learning for spatial branching: an algorithm selection approach". In: *INFORMS Journal on Computing* 35.5 (2023), pp. 1024–1043.

[Gor22] B. L. Gorissen. "Interior point methods can exploit structure of convex piecewise linear functions with application in radiation therapy". In: *SIAM Journal on Optimization* 32.1 (2022), pp. 256–275.

[GK20] B. Grimstad and B. R. Knudsen. "Mathematical programming formulations for piecewise polynomial functions". In: *Journal of Global Optimization* 77.3 (2020), pp. 455–486.

[GKK22] A. Gupte, A. M. Koster, and S. Kuhnke. "An adaptive refinement algorithm for discretizations of nonconvex QCQP". In: *20th International Symposium on Experimental Algorithms: SEA 2022*, ed. by C. Schulz and B. Uçar. Vol. 233. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl Publishing, 2022, 24:1–24:14.

[Hor86] R. Horst. "A general class of branch-and-bound methods in global optimization with some new approaches for concave minimization". In: *Journal of Optimization Theory and Applications* 51.2 (1986), pp. 271–291.

[HV22] J. Huchette and J. P. Vielma. "Nonconvex piecewise linear functions: advanced formulations and simple modeling tools". In: *Operations Research* (2022).

[KFN04] A. B. Keha, I. R. de Farias, and G. L. Nemhauser. "Models for representing piecewise linear cost functions". In: *Operations Research Letters* 32.1 (2004), pp. 44–48.

[KFN06] A. B. Keha, I. R. de Farias, and G. L. Nemhauser. "A branch-and-cut algorithm without binary variables for nonconvex piecewise linear optimization". In: *Operations Research* 5 (2006), pp. 847–858.

[Kes+04] P. Kesavan, R. J. Allgor, E. P. Gatzke, and P. I. Barton. "Outer approximation algorithms for separable nonconvex mixed-integer nonlinear programs". In: *Mathematical Programming* 100 (2004), pp. 517–535.

[AS00] F. A. Al-Khayyal and H. D. Sherali. "On finitely terminating branch-and-bound algorithms for some global optimization problems". In: *SIAM Journal on Optimization* 10.4 (2000), pp. 1049–1057.

[KRT22] J. Kim, J.-P. P. Richard, and M. Tawarmalani. *Piecewise polyhedral relaxations of multilinear optimization*. Preprint. 2022. Optimization Online: https://optimization-online.org/?p=19069.

[KM20] L. Kong and C. T. Maravelias. "On the derivation of continuous piecewise linear approximating functions". In: *INFORMS Journal on Computing* 32.3 (2020), pp. 531–546.

[LSW08] S. Leyffer, A. Sartenaer, and E. Wanufelle. "Branch-and-refine for mixed-integer nonconvex global optimization". Preprint ANL/MCS-P1547-0908. Mathematics and Computer Science Division, Argonne National Laboratory, 2008, pp. 40–78.

[LS13] M. Locatelli and F. Schoen. *Global optimization: Theory, algorithms, and applications*. Vol. MO15. MOS-SIAM Series on Optimization. SIAM, 2013.

[LHH23] B. Lyu, I. V. Hicks, and J. Huchette. *Building formulations for piecewise linear relaxations of nonlinear functions*. Preprint. 2023. arXiv: 2304.14542 [math.OC].

[MS04] T. L. Magnanti and D. Stratila. "Separable concave optimization approximately equals piecewise linear optimization". In: *Integer Programming and Combinatorial Optimization: IPCO 2004*, ed. by D. Bienstock and G. Nemhauser. Vol. 3064. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2004, pp. 234–243.

[MM57] H. M. Markowitz and A. S. Manne. "On the solution of discrete programming problems". In: *Econometrica* 25.1 (1957), pp. 84–110.

[Mey76] R. R. Meyer. "Mixed integer minimization models for piecewise-linear functions of a single variable". In: *Discrete Mathematics* 16.2 (1976), pp. 163–171.

[Nag+19] H. Nagarajan, M. Lu, S. Wang, R. Bent, and K. Sundar. "An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs". In: *Journal of Global Optimization* 74.4 (2019), pp. 639–675.

[NP09] J. M. Natali and J. M. Pinto. "Piecewise polynomial interpolations and approximations of one-dimensional functions through mixed integer linear programming". In: *Optimization Methods & Software* 24.4-5 (2009), pp. 783–803.

[PUK20] M. Posypkin, A. Usov, and O. Khamisov. "Piecewise linear bounding functions in univariate global optimization". In: *Soft Computing* 24.23 (2020), pp. 17631–17647.

[Reb16] S. Rebennack. "Computing tight bounds via piecewise linear functions through the example of circle cutting problems". In: *Mathematical Methods of Operations Research* 84.1 (2016), pp. 3–57.

[RK15] S. Rebennack and J. Kallrath. "Continuous piecewise linear delta-approximations for univariate functions: computing minimal breakpoint systems". In: *Journal of Optimization Theory and Applications* 167.2 (2015), pp. 617–643.

[RK20] S. Rebennack and V. Krasko. "Piecewise linear function fitting via mixed-integer linear programming". In: *INFORMS Journal on Computing* 32.2 (2020), pp. 507–530.

[SS98] J. P. Shectman and N. V. Sahinidis. "A finite algorithm for global minimization of separable concave programs". In: *Journal of Global Optimization* 12 (1998), pp. 1–36.

[She01] H. D. Sherali. "On mixed-integer zero-one representations for separable lower-semicontinuous piecewise-linear functions". In: *Operations Research Letters* 28.4 (2001), pp. 155–160.

[Sol71] R. M. Soland. "An algorithm for separable nonconvex programming problems ii: nonconvex constraints". In: *Management Science* 17.11 (1971), pp. 759–773.

[SSN22] K. Sundar, S. Sanjeevi, and H. Nagarajan. "Sequence of polyhedral relaxations for nonlinear univariate functions". In: *Optimization and Engineering* 23.2 (2022), pp. 877–894.

[TS04] M. Tawarmalani and N. V. Sahinidis. “Global optimization of mixed-integer nonlinear programs: a theoretical and computational study”. In: *Mathematical programming* 99.3 (2004), pp. 563–591.

[TV12] A. Toriello and J. P. Vielma. “Fitting piecewise linear continuous functions”. In: *European Journal of Operational Research* 219.1 (2012), pp. 86–95.

[Tuy16] H. Tuy. *Convex Analysis and Global Optimization*. 2nd edition. Vol. 110. Springer Optimization and its Applications. Springer Cham, 2016.

[TH88] H. Tuy and R. Horst. “Convergence and restart in branch-and-bound algorithms for global optimization. Application to concave minimization and DC optimization problems”. In: *Mathematical Programming* 41.1 (1988), pp. 161–183.

[VAN10] J. P. Vielma, S. Ahmed, and G. Nemhauser. “Mixed-integer models for nonseparable piecewise-linear optimization: unifying framework and extensions”. In: *Operations Research* 2 (2010), pp. 303–315.

[VKN08] J. P. Vielma, A. B. Keha, and G. L. Nemhauser. “Nonconvex, lower semicontinuous piecewise linear optimization”. In: *Discrete Optimization* 5.2 (2008), pp. 467–488.

[VN09] J. P. Vielma and G. L. Nemhauser. “Modeling disjunctive constraints with a logarithmic number of binary variables and constraints”. In: *Mathematical Programming* 128.1-2 (2009), pp. 49–72.

[WR22] J. A. Warwicker and S. Rebennack. “A comparison of two mixed-integer linear programs for piecewise linear function fitting”. In: *INFORMS Journal on Computing* 34.2 (2022), pp. 1042–1047.

[WR24] J. A. Warwicker and S. Rebennack. “Efficient continuous piecewise linear regression for linearising univariate non-linear functions”. In: *IISE Transactions* 0.0 (2024), pp. 1–15.

[WB14] A. Wechsung and P. I. Barton. “Global optimization of bounded factorable functions with discontinuities”. In: *Journal of Global Optimization* 58.1 (2014), pp. 1–30.

[YV13] S. Yldz and J. P. Vielma. “Incremental and encoding formulations for mixed integer programming”. In: *Operations Research Letters* 41.6 (2013), pp. 654–658.

[ZF12] M. Zhao and I. R. de Farias. “The piecewise linear optimization polytope: new inequalities and intersection with semi-continuous constraints”. In: *Mathematical Programming* 141.1-2 (2012), pp. 217–255.